**MASTER EN SCIENCES INFORMATIQUES**

# Méthode de sélection d'un langage de quatrième génération

Laurent, Jean-Philippe

*Award date:*
1990

*Awarding institution:*
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
NAMUR
*Institut d'informatique*

# METHODE DE SELECTION D'UN LANGAGE DE QUATRIEME GENERATION

Jean-Philippe Laurent

Mémoire présenté en vue de
l'obtention du titre de
Licencié et Maître en Informatique

1989-1990

Nos premiers remerciements s'adressent à Monsieur Jean-Luc Hainaut pour les conseils qu'il nous a prodigués dans un domaine aussi trouble que celui de la quatrième génération.

Nous souhaitons également remercier les personnes suivantes, qui ont, de près ou de loin, aidé à la réalisation de ce travail:

**Gécamines:** Messieurs Bourgeois, Druck, Graas, Kabangu, Kabamba, S'jongers, Trompette et Umba, pour leur accueil et le temps qu'ils ont accepté de nous consacrer.

**Logistique:** Monsieur et Madame André Hediger sans lesquels le stage n'aurait pu se réaliser.

Monsieur Didier Laurent pour ses nombreux conseils.

Messieurs Jean-Luc, Guy et Edgar Robins pour le matériel qu'ils ont accepté de mettre à notre disposition.

Monsieur Etienne Crochelet pour ses conseils quant à l'utilisation du L4G FoxBase.

# Table des matières

# 1. INTRODUCTION

La raison principale de migration vers les langages de quatrième génération (L4G) tient en deux mots: résorber le retard de développement [HENN.87]. Pour ce faire, deux approches sont utilisées:

- augmenter la productivité des informaticiens et diminuer la charge de maintenance, via les L4G **opérationnels**

- faire participer l'utilisateur final au développement des applications qu'il demande, via les L4G **décisionnels**

Globalement, l'informatique opérationnelle visera une amélioration de la productivité, de l'efficacité des tâches et décisions opérationnelles. Les applications opérationnelles sont lourdes, stables, planifiées

Alors que l'informatique stratégique (décisionnelle) visera à faciliter la découverte d'opportunités, de menaces, ... bref, à aider la prise de décision, voire à améliorer sa qualité. Les applications décisionnelles sont légères, ponctuelles, imprévisibles et souvent urgentes.

Pour atteindre ces objectifs, les L4G doivent être parfaitement **adaptés** à l'environnement dans lequel ils vont être utilisés; cette adaptation est d'ailleurs beaucoup plus importante pour un L4G que pour un L3G, principalement parce que les L4G sont d'usage moins général.

*"...(les L4G) ne peuvent être utilisés pour créer n'importe quel type d'application. Ils ne sont pas d'usage général. C'est le prix que nous avons à payer pour les grandes améliorations de productivité qu'apportent les L4G. Dans ce cas, nous devons sélectionner le langage adapté à l'application. Cela peut choquer quelques programmeurs puristes. Mais il est très important de comprendre que les langages à portée limitée permettent à l'utilisateur d'obtenir les résultats qu'il veut rapidement, alors que les L3G classiques ne le permettaient pas."* [Mart.86.a]

Les L4G, tant orientés développeurs qu'orientés utilisateurs finals, n'atteignent leurs objectifs qu'au prix d'une restriction de leur portée d'utilisation. En effet, deux concepts fondamentaux des L4G [d'après BOUT.84] les rendent particulièrement dépendants du contexte d'utilisation: la facilité d'emploi (surtout pour les L4G orientés utilisateurs finals), et la non-procéduralité.

## La facilité d'emploi

Les L4G destinés aux utilisateurs finals doivent se mettre à leur portée, les placer dans un environnement le plus familier possible, se rapprocher de l'univers de leur tâche. Rapprocher l'outil des préoccupations de l'utilisateur permet en effet de limiter l'effort de formation et de programmation qui lui est demandé. Revers de la médaille: un L4G ne conviendra qu'à un type de tâche, à une catégorie d'utilisateurs. Plus un L4G sera simple d'emploi (en se rapprochant de l'utilisateur), plus il restreindra son domaine d'application. Citons [ROUX.88]:

*"...aucun produit du marché n'offre de manière satisfaisante l'ensemble des dix classes de fonctions (d'un L4G pour utilisateur final) et probablement aucun produit ne le fera jamais, ce qui en fait n'est pas grave car, à mes yeux, il n'est pas souhaitable qu'un tel produit existe car, voulant couvrir trop de fonctions, il ferait certainement trop de compromis pour bien les couvrir toutes...".*

## La non-procéduralité

Les L4G doivent aussi améliorer la productivité de la programmation, en "pré-mâchant" le travail, en proposant un langage de haut niveau, souvent non-procédural. Ils tentent de diminuer la tâche de programmation en se rapprochant le plus possible des spécifications. Cependant, la non-procéduralité n'est un gage de facilité et de rapidité de développement que si l'outil est adapté:

*"Etant donné l'architecture courante des O4G (outils de quatrième génération), un utilisateur peut gagner du temps de développement si le problème correspond aux hypothèses des facilités non-procédurales pré-définies de l'outil. Si le problème n'est pas du type pour lequel l'outil a été conçu, l'utilisateur peut payer des pénalités en développement et performances. Dans ce cas, la programmation conventionnelle est une meilleure alternative."*

Ou encore:

*"Si votre O4G ne vous permet pas de résoudre un problème directement par un outil non-procédural (ex: report generator inadapté), vous devrez écrire un programme dans un langage très étrange - certains d'entre eux se situant très en-deçà des possibilités des L3G."* [MISR.88]

Les L4G, tant pour les développeurs que pour les utilisateurs finals, doivent donc prendre en compte les contraintes imposées par leur environnement, même si toutes ces contraintes ne s'appliquent pas avec la même force sur ces deux types d'outils.

De plus, *"dans bien des cas, un L4G est plus un moteur de changement organisationnel qu'un outil technique."* [HAIN.88] L'introduction des L4G

bouleverse les habitudes de travail. D'où l'intérêt de bien évaluer les contraintes, les risques et les conséquences que ces outils auront sur l'environnement organisationnel.

Si l'on veut profiter de la puissance d'un L4G, il doit être parfaitement adapté à l'utilisation qu'on veut en faire. Il semble donc raisonnable de prendre pour postulat de départ que la **productivité** d'un L4G dépend de son **adaptation** à l'environnement dans lequel il est utilisé (méthodes de développement utilisées, domaine d'application visé, rôle des utilisateurs et des informaticiens,...).
D'où l'importance cruciale que revêt pour une entreprise le choix du ou des L4G à acquérir.

Le **but de ce mémoire** est d'offrir une méthode de sélection d'un L4G qui soit générale, cohérente et praticable. Générale, pour permettre de choisir un L4G dans tous les cas de figure possible; cohérente en s'articulant autour de structures et concepts théoriques solides; praticable en se basant, non pas sur une comparaison théorique abstraite, mais sur les besoins réels des environnements d'utilisation possible.

## But du mémoire:

De nombreuses comparaisons ou classements des L4G existent déjà. Quelle est la spécificité de l'approche proposée ici, par rapport à ces études ?

Pour la justifier, il suffit de partir des critiques que l'on peut émettre à l'égard des analyses "classiques", que l'on peut classer en quatre catégories:

- les définitions, classements, typologies, qui tentent de mettre de l'ordre dans la jungle de l'offre en matière de L4G, appellation très souvent usurpée pour des raisons de marketing
[BOUT.84, ROUX.88, MART.85, HAIN.88]

- les analyses théoriques qui tentent d'évaluer l'impact des L4G sur les méthodes d'analyse et de développement, sur la gestion de la ressource informatique,...
[LAUR.89, HENN.87, HAIN.88, MART.85]

- les comparaisons générales de L4G dont on évalue les fonctionnalités et caractéristiques, qui débouchent généralement sur un classement global
[BOUT.84, GCM.86, MART.86 a et b, ROUX.88, SVM, BYTE]

- les comparaisons de L4G sur un point précis: les gains de productivité, les performances,... Il s'agit là d'études longues et très fouillées.
[MISR.88, VERN.88, JALI.89]

Les principales **critiques** formulées à l'encontre de ces analyses sont les suivantes:

- les comparaisons et panoramas sont souvent **incomplets**: ils ne prennent en compte qu'une partie de l'offre en L4G (difficultés pour rassembler tous les L4G disponibles dans un environnement donné)

- ils **vieillissent** très vite: toute nouveauté (nouvel L4G ou nouvelle version) est susceptible de perturber le classement établi. Si l'on considère qu'un L4G est amélioré chaque année (surtout en micro-informatique), une étude vieille de plus d'un an devra être utilisée en surveillant de près les numéros de versions.

- les classements, qui se veulent tous "objectifs", sont très dépendants des méthodes et conditions de test, des opinions pré-conçues des testeurs, ... En outre, leur base est souvent différente; comment dès lors pouvoir en faire une synthèse fiable et cohérente ? Les **standards** de comparaison n'existent pas.

- les comparaisons sont toutes basées sur une évaluation des fonctionnalités des L4G; elles partent de l'outil lui-même, et pas d'une analyse des besoins des utilisateurs. Cela a pour conséquence de reporter

sur l'analyse le **manque de structure**, le fouillis qui caractérise la quatrième génération. Il existe bien des tentatives de définition des L4G, mais, face à leur grande diversité, elles doivent prendre beaucoup de recul et ne conserver que des critères généraux très flous (convivialité, non-procéduralité, intégration, ...), peu praticables dans une analyse concrète.

- chaque nouvel L4G offre des approches et solutions originales, rendant caduques certaines fonctionnalités traditionnelles. Plus que les fonctionnalités offertes, c'est la **philosophie** du L4G qui permet de les caractériser. Tenter d'élaborer un classement général et objectif, c'est s'enfermer dans un canevas forcé et artificiel, gommant les spécificités. On objectera que c'est le propre de toute comparaison d'imposer une structure de référence. Certes, se passer d'une structure empêcherait de tirer aucune conclusion pratique, mais l'appliquer trop strictement masquerait l'essentiel dès qu'il s'agit d'un contexte aussi riche et disparate que les L4G.

- beaucoup de ces analyses, surtout les plus anciennes, définissent les L4G comme des outils de développement d'applications par les utilisateurs finals. Le temps a passé depuis et les expériences réussies dans cette voie ne sont pas légion ! Il semble que la programmation, aussi légère soit-elle, d'une application complexe requière toujours une méthode et une organisation rigoureuse.

Il n'en reste pas moins que les études théoriques [LAUR.89] , les évaluations pratiques [comparaison des performances JALI.89] et les comparaisons de L4G sont indispensables. Elles demandent un travail considérable, qu'il eût été vain de vouloir reprendre dans le cadre de la présente étude. C'est pourquoi, malgré leurs faiblesses et leur incomplétude, nous proposons de les utiliser sans les remanier. Elles sont indispensables pour une des étapes de la méthode proposée ici, où elles seront d'ailleurs abondamment utilisées [chapitre 5: pré-sélection].

Une analyse basée sur les seuls besoins des utilisateurs serait incomplète: l'analyse des besoins de l'utilisateur permet de mettre l'accent sur certaines caractéristiques souhaitées, mais n'envisage pas l'ensemble des qualités indispensables, qui sont en quelque sorte sous-entendues. C'est pourquoi le point 3 "Critères généraux" du chapitre 4 tente de réintroduire la notion de comparaison des grandes fonctions des L4G sur base de leurs caractéristiques fondamentales (convivialité, non-procéduralité,...). Mais c'est en tenant compte des critiques émises ici: une comparaison ne peut se faire que sur base de critères de référence communs à tous les L4G, et donc très généraux; il faut dès lors garder à l'esprit que la généralité de ces critères empêche tout classement rigide et artificiel, qui ferait l'impasse sur l'originalité des différentes approches proposées par les L4G.

Face à ces critiques, ce mémoire propose de prendre du recul par rapport à ces analyses ponctuelles, et de les replacer dans une structure cohérente, centrée, non plus sur des fonctionnalités trop différentes d'un L4G à l'autre, mais sur les besoins de l'utilisateur et les contraintes de l'environnement organisationnel. Il va tenter de mieux structurer les paramètres de choix d'un L4G en les replaçant dans un contexte plus large et plus stable: celui de l'utilisateur. La méthode proposée ici se veut stable et générale; tous les paramètres de décision trop spécifiques ou susceptibles d'évolution sont écartés. On y accédera via des analyses, essais, comparaisons, classements,... extérieurs au présent cadre d'étude.

## Limites du mémoire:

Dans un souci de généralité et de stabilité dans le temps, la méthode proposée ici s'est fixé des limites. En effet, dans l'analyse des critères, on restera à un niveau suffisamment **général** pour pouvoir englober tous les types de L4G, existants et à venir. Les qualités rendant un L4G convivial, par exemple, sont susceptibles d'évoluer avec de nouvelles approches. Une idée originale dans ce domaine peut obliger à revoir les standards qui permettaient jusque là d'attribuer le label de convivialité à un L4G. Plutôt que de s'enfermer dans

une check-list de paramètres précis que devrait posséder une fonction particulière d'un L4G, nous avons préféré nous dégager des caractéristiques spécifiques, pour nous arrêter aux **grandes classes de fonctionnalités** [chapitre 4].

Nous pensons qu'une évaluation ne peut se limiter à cocher des cases dans des tables de paramètres, malgré le sentiment d'objectivité que peut procurer une telle méthode. Dans les procédures de sélection que nous avons pu examiner, le véritable moteur de la décision était la libre appréciation, souvent plus intuitive qu'objective. Donner à un choix intuitif une apparence d'objectivité prend plus de temps que la décision elle-même; est-ce bien utile, si ce n'est pour se justifier vis-à-vis de supérieurs ? Le choix d'un L4G doit être franchement basé sur l'appréciation du décideur, qui utilisera pour ce faire tous les moyens qu'il juge utile, y compris les check-lists, par ailleurs très complètes, que l'on peut trouver dans [MART.85].

Ce choix de dégager la **méthode** de sélection des **moyens** qu'elle utilise permet de la laisser ouverte à l'évolution. La méthode restera stable, seuls les moyens de mesure précis devront s'adapter. Le but de ce mémoire n'est pas d'établir une comparaison de L4G précis, valable à un moment donné, mais de proposer une méthode générale de sélection.

## Plan succinct:

Dans la première partie de ce mémoire, nous proposerons une méthode de choix du L4G le mieux adapté à un environnement d'utilisation déterminé, et partant, le plus productif.
En gros, l'environnement va imposer des contraintes générales [chapitre 3], que l'on transformera en critères [chapitre 4] qu'un L4G devra vérifier pour être sélectionné [chapitre 5].

Dans la seconde partie, nous tenterons d'étayer ces critères par des expérimentations.

Nous allons commencer par brosser un tableau des L4G et de leurs caractéristiques; nous ferons également le point sur les méthodes classiques de sélection de langages.

# 2. ETAT DE L 'ART

## 2.1. Les L4G

### 1. définition

Un langage de quatrième génération (L4G) est un outil d'informatique de gestion offrant des fonctions conviviales, non-procédurales, intégrées et efficaces [BOUT.84]. Son objectif est d'alléger (voire supprimer) la tâche de programmation en la rapprochant des spécifications données par l'homme.
Dès que l'on veut préciser cette définition, elle ne reflète plus la totalité de la "famille L4G" car elle rentre dans une catégorie trop précise.

### 2. les générations

La tradition veut que les langages informatiques soient classés en générations chronologiques, dans une ligne d'évolution dont la tendance principale est l'éloignement progressif du langage machine pour se rapprocher du langage humain.

La **première génération** (1950): le langage machine, dont les primitives sont tout simplement les instructions du processeur. Ces instructions, sous forme de code binaire, doivent être rangées à leur adresse physique dans la mémoire.

La **deuxième génération** (1955): l'assembleur, dont les primitives sont toujours les instructions du processeur, mais représentées sous forme mnémonique. L'allocation des emplacements physiques en mémoire est prise en charge par le programme d'assemblage.

La **troisième génération** (1960): les langages de haut niveau, prenant du recul par rapport aux instructions du processeur en les regroupant en primitives de haut niveau, plus lisibles par l'homme et plus indépendantes du processeur.

La **quatrième génération** (1975): langages voulant diminuer la charge de programmation, par la prise en charge immédiate des spécifications données par l'analyse.

La **cinquième génération** (1980): langages d'intelligence artificielle, proposant de rendre la machine capable de raisonnements humains, dans certains domaines de connaissance bien précis.

De grandes **tendances** peuvent être dégagées de cette évolution:

- on s'éloigne de la machine pour se rapprocher des spécifications données par l'homme: la facilité d'emploi s'accroît, le langage change de niveau (passage progressif du procédural des trois premières générations au non-procédural pour les 4ème et 5ème générations)

- en s'éloignant de la machine, on perd du contrôle sur elle; le pilotage devient moins précis, moins souple; le champ d'application se rétrécit

- le code nécessaire pour "programmer" une application se réduit, et avec lui le temps de développement (l'effort de développement croît linéairement avec le volume de code, indépendamment du langage utilisé: [JALI.89])

Ces générations ne sont pas exclusives: certains outils se trouvent à la croisée de deux générations (le C et ses possibilités de contrôle proches de l'assembleur, par exemple). En outre, l'apparition d'une nouvelle génération n'implique pas la disparition de la précédente. C'était déjà vrai pour l'assembleur qui a continué sa route à côté des L3G; ça l'est plus encore pour

les L4G et L5G dans la mesure où leur domaine d'application restreint les rend complémentaires des L3G, qui restent incontournables pour bon nombre d'applications.

Cette lignée donne l'impression que l'évolution des langages est continue; or, les quatrième et cinquième générations marquent une rupture profonde par rapport aux précédentes. Jusque là, le moteur de l'évolution se basait sur la machine: on agrégeait les instructions du processeur en primitives de plus en plus évoluées, mais les langages ainsi produits étaient toujours basés sur une logique orientée-machine. Les langages de quatrième et cinquième générations, eux, se basent sur les spécifications données par l'homme, et s'efforcent de lui cacher au maximum les contraintes liées à la machine.

# 3. problèmes liés à la troisième génération

### *l'informatique décisionnelle*

*"La priorité à l'informatique opérationnelle, lourde par nature, étouffe les demandes de développement d'applications légères, urgentes, ponctuelles et non planifiées. Or celles-ci émanent typiquement des niveaux tactiques et stratégiques de l'organisation, c'est-à-dire qu'elles constituent ce qu'on a nommé l'informatique décisionnelle."* [HAIN.88]

Pour développer de petites applications ponctuelles demandées par les utilisateurs, les informaticiens utilisent les mêmes méthodes que pour les projets lourds (paie, stocks,...); elles sont totalement inadaptées à ces demandes légères et imprévisibles, qui doivent être satisfaites très vite.

### *La maintenance*

*"L'informatique se maintient, mais ne progresse plus"* [HAIN.88] Elle consacre l'essentiel de ses forces vives à la maintenance (80% selon [MART.85]); les demandes des utilisateurs s'accumulent, le développement d'applications nouvelles s'enlise,... Les méthodes de développement classiques sur L3G sont lourdes et fastidieuses, si bien qu'elles sont rarement appliquées

avec tout le soin nécessaire. Utilisées correctement, elles seraient certes longues et coûteuses, mais permettraient au moins de faciliter la maintenance. Cet avantage trop théorique est hélas rarement obtenu, puisque c'est précisément au niveau de la documentation qu'on opère des "raccourcis" pour accélérer le développement.

Plus une application est ancienne, plus elle devient inadaptée et difficile à maintenir. Il n'est pas rare qu'une application de paie du personnel vieille de 15 ans soit toujours en exploitation ! La plupart de ces applications sont devenues des monstres incompréhensibles car, même si leur conception originale était claire, la succession d'équipes de maintenance venues à leur chevet a fait disparaître toute structure cohérente. La maintenance d'une telle application s'apparente plus à du "reverse engineering": devoir comprendre ce que fait le programme à partir de ses seules entrées/sorties. La décision de réécrire totalement des applications aussi énormes demande beaucoup de courage: où trouver les moyens nécessaires pour redévelopper les applications opérationnelles, alors que le CTI est déjà totalement saturé ? On préférera généralement attendre jusqu'à leur dégradation complète. Martin estime que le coût de reprogrammation de toutes les applications développées par L3G dans le monde atteindrait 1000 milliards de dollars. Autant dire que cela ne se fera pas du jour au lendemain, et que Cobol, qui détient près de 70% du marché, n'est pas près de disparaître ! Si cette catastrophe ne peut être réparée immédiatement, du moins peut-on éviter qu'elle ne s'aggrave. C'est pourquoi on adopte de plus en plus couramment une solution consistant à décider que tout développement d'application nouvelle se fera avec des outils qui rendront la maintenance plus facile (L4G, outils de génie logiciel,...), à l'exclusion de tout L3G traditionnel.

### *la productivité du développement*

Le retard de développement officiel atteint 2 à 4 ans [MART.85]. A cela s'ajoute la plupart des besoins décisionnels des utilisateurs qui, vu le "backlog" déjà considérable, ne sont pas repris. C'est d'ailleurs pour cette raison que, lors de l'installation d'un L4G décisionnel, les utilisateurs s'attaquent d'abord à ces demandes non-satisfaites, et les informaticiens sont tout étonnés de

constater que le backlog officiel, qui n'en tient pas compte, ne diminue pas. Nous avons vu que la charge de maintenance de l'énorme parc d'applications installées était une donnée incontournable, qui pèsera encore longtemps sur le monde de l'informatique. Mais il ne faut pas que cela empêche le développement d'applications nouvelles, qui devra se faire avec les ressources humaines limitées qu'impose la maintenance. C'est dans cette optique que l'utilisation de L4G se généralise: permettre de développer des applications plus rapidement et avec moins de programmeurs.

## 4. caractéristiques des L4G

Les éléments-clé des L3G étaient les instructions, groupées en procédures. Les éléments-clé des L4G sont les objets manipulés par l'application: relations de la base de données, formulaires et rapports de l'interface (ou dialogue) et procédures ou formules représentant les traitements. La description de ces objets est centralisée dans le dictionnaire des données, véritable centre nerveux des L4G.

Pour manipuler ces objets, les L4G offrent quatre **familles de fonctions**:

- les fonctions liées à la définition et à la gestion de données (composante de base de données): administration de la structure de données, consultation et gestion des données.

- les fonctions de présentation des données (composante de dialogue): formulaires de saisie, rapports, graphiques,...

- les fonctions de traitement des données (composante de traitement): processeur de procédures écrites dans le langage procédural du L4G, générateurs d'application, processeur de modèles (programmation non-algorithmique comme dans les tableurs), processeurs statistiques, moteurs d'inférence,...

- les fonctions d'échange avec l'environnement (composante d'échange): communication, conversion, distribution,...

- Les L4G les plus récents (voir point 6: L4G avancés) offrent en outre la possibilité d'intégrer la dynamique de l'application au sein même des objets du dialogue et de la BD (composante de pilotage).

Nous analyserons plus en détail ces composantes aux chapitres 3 et 4.

Pour faciliter l'utilisation de ces fonctions, les L4G privilégieront la convivialité, la non-procéduralité, et l'intégration, qui sont leurs principaux atouts.

### *la convivialité*

*"La convivialité d'un outil logiciel, c'est littéralement la possibilité de vivre avec lui, avec tout ce que cela peut impliquer en termes de confiance, dialogue, fidélité, disponibilité, durabilité, aisance et même plaisir, etc."* [BOUT.84]

### *la non-procéduralité*

La programmation procédurale impose de spécifier pas à pas la séquence d'opérations élémentaires à effectuer pour arriver au but recherché. La programmation non-procédurale prend de la hauteur par rapport au détail des opérations; il suffit de spécifier le but à atteindre, sans se préoccuper du cheminement opératoire. Dans le cas des L4G, l'utilisateur n'aura qu'à spécifier de façon déclarative l'objectif qu'il veut atteindre, l'outil se chargera lui-même des détails de mise en oeuvre. En L4G, l'utilisateur pilote les opérations; en L3G, il doit les programmer.

La non-procéduralité des L4G se retrouve principalement dans les outils suivants: design des formulaires de saisie, des rapports, de la structure BD et ses contraintes d'intégrité, spécification de formules, de critères de recherche, de tri, écriture de procédures, ...
En outre, les L4G sont basés sur des SGBD relationnels, considérés comme non-procéduraux en comparaison des BD hiérarchiques.

*l'intégration*

Les fonctions offertes par les L4G se retrouvaient auparavant dans des outils différents. Leur grand mérite est de les rassembler en une structure cohérente (le dictionnaire des données) et dans un langage de manipulation unique.

Ces caractéristiques et leurs implications sur l'outil seront mieux analysées au chapitre 4.

# 5. solutions proposées par la quatrième génération

Pour résoudre les problèmes liés à la troisième génération, deux **voies de recherche** sont suivies actuellement:

- la lignée des L4G vise la productivité immédiate. Elle préconise de rapprocher l'outil des spécifications de l'application, afin d'automatiser le développement, au détriment de la réutilisabilité et de la productivité à long terme (amélioration de la productivité immédiate par rapport aux L3G : 1000% [BOUT.84]). Cette lignée, lancée par les fournisseurs de soft, est fort pragmatique; il lui manque encore une bonne assise théorique, principalement au niveau de la méthode de développement à suivre.

- la lignée des outils de génie logiciel (ou outils CASE : Computer Aided Software Engineering) vise la productivité à long terme. Elle préconise de favoriser la réutilisabilité des applications, au détriment du temps de développement et de la productivité immédiate (amélioration de la productivité immédiate par rapport aux L3G : 10 à 50% [BOUT.84]). Ces outils facilitent l'écriture et la vérification des programmes, mais ne les font pas disparaître. Ils tentent au contraire de les modulariser, afin de les rendre généraux et réutilisables. Cette lignée fait surtout partie du domaine de la recherche théorique (universitaire); il lui manque encore une implantation solide dans les entreprises.

Nous englobons dans cette catégorie les L3G de haut niveau, privilégiant la modularité (Ada, Modula-2,...), les langages orientés-objet, les outils d'écriture automatique de programmes à partir de spécifications rigoureuses, ainsi que les "accessoires" chargés de faciliter une étape précise du développement: outils de debugging, de production automatique de la documentation, de validation d'algorithmes ou d'assertions, ...

Les L4G proposent d'améliorer la productivité du Centre de Traitement Informatique (CTI) de deux manières:

- augmenter la productivité du développement et diminuer la charge de maintenance, via les L4G opérationnels

- décharger le CTI en offrant à l'utilisateur final les moyens de développer lui-même certaines applications, via les L4G décisionnels

### *l'informatique décisionnelle*

La convivialité, la non-procéduralité, et l'intégration des fonctions qu'offrent les L4G rendent accessible aux utilisateurs finals la puissance de traitement informatique, qu'ils n'utilisaient autrefois que par informaticien interposé.

La diffusion au sein de l'entreprise d'outils décisionnels pour utilisateurs finals ne peut se concevoir sans une structure d'assistance solide: l'**infocentre**. L'infocentre est une structure d'assistance aux utilisateurs finals, composée d'informaticiens et d'utilisateurs expérimentés, distincte du CTI (centre de traitement informatique), chargée plusieurs **missions**:

- aider les utilisateurs lors de la sélection, de l'installation et de l'utilisation de matériel et de logiciel (souvent des micro-ordinateurs et des L4G).

- assurer la formation et l'assistance aux utilisateurs

- promouvoir une certaine standardisation dans le matériel et les logiciels; par exemple, l'infocentre n'assistera les utilisateurs de tableurs que s'ils choisissent Lotus123 ou Quattro sur PC-compatible. Cela évitera d'avoir une architecture micro, et une gamme de logiciels, hétérogène et incapable d'échanger ses données.

- maintenir un bon niveau de sécurité quant à l'accès aux réseaux

- maintenir l'intégrité de la base de données centrale (sur mainframe), en fournissant aux utilisateurs des extraits quotidiens des données qui les intéressent.

- maintenir un bon niveau de performance machine; si certains outils décisionnels sont situés sur le mainframe, il faut empêcher les utilisateurs d'effectuer des requêtes absurdes et coûteuses en temps machine.

C'est un compromis entre les desideratas des utilisateurs et ceux des informaticiens: les premiers sont satisfaits de se voir aidés rapidement, ce à quoi le CTI ne les avait pas habitués; les seconds seront soulagés de toute l'informatique décisionnelle, tout en étant assurés que les utilisateurs ne seront pas laissés à eux-mêmes (problèmes de standardisation, performances,...). [LESU.89.b]

### *la productivité du développement*

L'idée de base des L4G est de découper l'application en objets visibles dès l'analyse fonctionnelle; ils évitent ainsi de devoir passer par un redécoupage en modules internes aux fonctionnalités abstraites, éloignées des fonctions et objets réels de l'application. Ce travail de recomposition des fonctions de l'analyse fonctionnelle prend beaucoup de temps; les L4G proposent tout simplement de s'en passer. En se rapprochant de la sorte des spécifications de l'analyse fonctionnelle, et en permettant de supprimer une bonne partie de la conception, ils font gagner beaucoup de temps de développement.

En ce qui concerne le gain de productivité que procurent les L4G, les auteurs n'ont pas le même avis; on peut distinguer les "enthousiastes" des "réalistes". Les premiers se basent sur des appréciations subjectives, et sur un consensus quasi-général qui veut que le gain de productivité moyen entre un bon L4G et Cobol soit de 10 contre 1 (développement 10 fois plus rapide avec les L4G). Les seconds [MISR.88, VERN.88] s'attachent à mesurer par des expérimentations le ratio réel L4G/Cobol, et arrivent à des conclusions beaucoup moins euphoriques: certains ratios sont même négatifs ! Il faut relativiser ces conclusions en comprenant que le temps de développement d'une application avec un nouvel L4G est toujours aggravé par la méconnaissance de cet outil, et par l'inadéquation des méthodes de développement utilisées. Paul Jalics, qui était arrivé à ces ratios déplorables en 1988, reconnaît d'ailleurs deux ans plus tard que "cela prend beaucoup plus de temps de développer un programme qui contient 10 à 20 fois plus de code qu'un autre qui accomplit la même tâche", et qu'en définitive, "le temps de développement doit probablement être une fonction monotone croissante de la taille du code" [JALI.90]. Ce revirement s'explique par le fait que, bien qu'aucune étude concrète ne parvienne à le prouver, tout le monde soit d'accord pour reconnaître que les L4G sont plus productifs. Le ratio de 4:1, par sa position intermédiaire entre les extrêmes, semble plus raisonnable; il est d'ailleurs souvent cité [VERN.88, HENN.87].

*la charge de maintenance*

En règle générale, plus un langage se rapproche des spécifications, plus le "décodage" d'un programme sera facile pour l'homme.

Plusieurs caractéristiques des L4G rendent la maintenance plus aisée:

- le dictionnaire des données qui centralise dans une structure claire et cohérente toutes les descriptions des objets composant l'application.

- la diminution du volume final de code

- le code non-procédural est plus lisible

Toutefois, l'éclatement des traitements et de la dynamique entre les différents objets de l'application peuvent rendre la maintenance et le debugging difficiles. En effet, l'état global atteint par le système à un moment donné n'est pas connu avec précision: contenu des variables, sélection courante des records, ... L'enchaînement des traitements est sous le contrôle du L4G; comment dès lors prévoir le comportement d'une procédure si l'état du système à l'entrée de cette procédure est imprévisible, puisqu'il dépend de ce que le L4G aura décidé de faire avant ? [voir chapitre 4]

# 6. typologie classique des L4G

On peut établir trois distinctions majeures entre les différents types de L4G [LAUR.89]:

### *le type d'utilisateur: L4G orientés utilisateur final ou développeur*

Un L4G destiné aux utilisateurs finals privilégiera la convivialité.

Un L4G destiné aux développeurs privilégiera la productivité.

### *le type d'application: L4G opérationnels ou décisionnels*

Un L4G destiné aux applications décisionnelles privilégiera l'accès aux données déjà stockées, leur présentation, bref, les outputs.

Un L4G destiné aux applications opérationnelles privilégiera le développement d'applications nouvelles (structuration, saisie, validation des données), bref, les inputs.

*le degré d'intégration: L4G traditionnels ou avancés*

Les L4G traditionnels sont, comme les L3G, centrés sur le concept de procédure; les traitements et la dynamique de l'application sont enfermés dans des modules.

Les L4G avancés sont centrés sur les objets de la BD et du dialogue. Les traitements et la dynamique de l'application sont éclatés au sein de ces objets. Ils disposent d'une fonction de pilotage qui leur permet de prendre en charge automatiquement la gestion et le contrôle de validité des enchaînements d'actions. Grâce à elle, *"le L4G examine à tout moment l'ensemble des opérations spécifiées de façon déclarative et effectue les opérations se rapportant à l'état dans lequel se trouve le système."* [LAUR.89]

La distinction habituelle entre L4G orienté développeurs et L4G orienté utilisateurs finals recoupe souvent celle qui est faite entre L4G opérationnels et décisionnels: on retrouvera de plus en plus des L4G décisionnels orientés utilisateur final et des L4G opérationnels orientés développeurs. L'expérience prouve en effet qu'il ne faut pas trop attendre des utilisateurs finals en matière de développement d'applications opérationnelles sérieuses.

# 7. problèmes liés aux L4G

*Problèmes de définition et de classement*

Tout nouveau produit lié au développement d'application doit se parer du label "Quatrième Génération" s'il veut avoir une chance de se vendre.
Le concept de "Quatrième Génération", lancé par Martin il y a une dizaine d'années, ne renferme rien de bien précis, laissant libre cours à l'imagination débordante du marketing des fournisseurs de soft. Les caractéristiques qui servent à les distinguer d'autres outils de développement sont trop vagues pour permettre de réglementer la "jungle des L4G". C'est un **fourre-tout** contenant des éléments aussi disparates que:

- des "vieux machins" ayant subi un lifting: L3G agrémentés de générateurs d'écrans, de possibilités de gérer une BD,..

- des progiciels paramétrables, individualisables

- des outils d'aide à la décision et d'informatique décisionnelle

- des systèmes de gestion de bases de données

- des générateurs d'application

- des langages de très haut niveau offrant des possibilités de programmation orientée-objets, pilotée par les événements, de programmation non-procédurale, d'intégration poussée des composants,...

Devant ce panorama, force est de constater que, si l'on en croit Martin, la quatrième génération regroupe tout ce qui se fait de neuf dans l'industrie du software , à la seule exception de l'intelligence artificielle, rangée dans une génération à part. La seule définition cohérente qu'on peut dès lors trouver pour la quatrième génération est "l'ensemble de toutes les approches visant à éliminer ou réduire la programmation". Vaste programme ! Or, si l'on se limite à la seule informatique opérationnelle, on peut distinguer **trois approches** très différentes:

- la lignée des L4G qui veut accélérer le développement.

- la lignée des outils de génie logiciel, qui veut faciliter la maintenance et la réutilisabilité des applications, plus que la rapidité de développement.

- la lignée des prociciels d'application, qui veut réduire le développement à la seule adaptation de l'outil à l'environnement d'utilisation. La panoplie des domaines couverts ne cesse de s'étoffer: gestion comptable, financière, commerciale, gestion de projets et ordonnancement, mais aussi automation, CAD/CAM, support de R&D, ...

La plupart des entreprises utiliseront plusieurs approches, choisissant leurs outils en fonction du degré d'expertise des utilisateurs, des domaines d'application,... La Gécamines, par exemple, a adopté la stratégie suivante en matière d'informatique opérationnelle:

- choisir de préférence un progiciel, pour autant qu'il réponde à 80% des besoins du domaine.

- si aucun progiciel n'est satisfaisant, développer l'application "in-house" à l'aide d'un L4G opérationnel (CA-Universe), à l'exclusion de tout L3G.

Les L4G les plus courants s'intéressent au développement de systèmes d'information de gestion classiques (opérationnels ou décisionnels). Il semble raisonnable de ne pas étendre la quatrième génération au-delà des systèmes permettant le développement d'applications. Un progiciel paramétrable ou un langage d'application ne sera considéré comme L4G que s'il permet de développer des applications personnalisées; la simple paramétrisation ou "customisation" d'un progiciel aux besoins du client nous semble insuffisante pour lui attribuer le label "L4G". Cependant, certains progiciels offrent à l'utilisateur des outils de génération d'applications décisionnelles puissants (SDT pour la package de gestion comptable Millenium Applications de Mc Cormack & Dodge).

Cette situation confuse est à rapprocher de celle qui prévalait au début des années 70 dans le domaine des langages de haut niveau, appelés maintenant L3G. Des 120 L3G qu'on pouvait répertorier en 1969 [MART.85], moins de 10 ont réellement survécu; ce n'était pas toujours les meilleurs, mais plutôt ceux qu'utilisaient les grandes entreprises. Des standards ont peu à peu fait leur apparition, permettant de structurer le marché. 20 ans après, on constate qu'un consensus s'est établi en ce qui concerne les méthodes de développement à utiliser, et les domaines d'application respectifs des différents L3G survivants. Cette évolution est courante dans les domaines où l'innovation est prépondérante, tels l'informatique; de nombreux autres exemples existent: les bases de données relationnelles (standard: SQL), les micro-ordinateurs (standard: IBM PC),... Chaque fois qu'une idée révolutionnaire apparaît, le

marché s'affole. La recherche s'active dans de nombreuses directions, l'imagination, l'audace et l'originalité règnent. Petit à petit, le marché tente de digérer l'innovation en favorisant l'apparition de standards. Les entreprises ont horreur d'investir dans des produits à l'avenir incertain; dès qu'un produit "sérieux" émerge, elles s'y réfugient et l'érigent en standard, obligeant le marché à se structurer en conséquence. Enfin vient la période de stabilité, pendant laquelle le marché vit de ses rentes.

Il sera aisé, d'ici dix ans, de tracer un historique semblable pour la quatrième génération. Actuellement, l'offre foisonnante et le chaos qui règnent encore sur le marché des L4G nous incitent à penser que nous sommes toujours dans la première phase de bouillonnement des idées. Toutefois, certains indices montrent que nous sommes en train d'en sortir: les absorptions de petits éditeurs se multiplient, l'offre de nouveaux produits s'essouffle, et certains L4G deviennent incontournables: par exemple, DB2 pour les mainframes, poussé par le rouleau compresseur d'IBM. En micro-informatique, la situation est plus confuse, principalement parce qu'IBM n'y joue pas (encore) son rôle de gendarme; on doit néanmoins reconnaître qu'un L4G pour micro-ordinateur se doit, d'une façon ou d'une autre, de permettre l'accès au format DBaseIII.

### *Problèmes de méthodologie de développement*

Plusieurs caractéristiques des L4G rendent les méthodes de développement traditionnelles inadaptées:

*"Les L4G se rapprochent plus du domaine de la spécification que de celui de la réalisation. D'autre part, les rapports entre les composants BD, dialogues et traitements s'y trouvent bouleversés: les traitements y apparaissent comme subordonnés aux composants de la base de donnée et des dialogues."* [LAUR.89].
Les L4G imposent un renversement complet des priorités. Pour les L3G, les traitements sont centraux, et la BD et le dialogue sont noyés dans le code des procédures. Au contraire, la BD et le dialogue deviennent le pivot des L4G, les traitements n'y sont que des appendices.

Autre différence avec les L3G: la dynamique de l'application était, chez ces derniers, enfermée dans un module (le "coordinateur"); chez les L4G, au contraire, elle est éclatée entre tous les objets qui composent l'application. Cela empêche de pouvoir développer la dynamique de façon autonome par rapport aux objets de l'application.

En outre, *"le fait qu'un objet de l'interface, le formulaire ou le rapport, intègre l'ensemble des actions liées aux changements d'état de ses objets constitutifs nous a également amené à remettre en question l'indépendance entre l'interface et les fonctionnalités de l'application."* [LAUR.89]

La méthode élaborée dans [LAUR.89] propose de recentrer l'analyse conceptuelle sur la décomposition d'application proposée par les L4G: on traduira les phases de l'analyse fonctionnelle en descriptions d'objets de la base de données (les relations), du dialogue (les formulaires), des traitements (procédures) et du pilotage (actions). Nous y reviendrons plus en détail lors de l'expérimentation [chapitre 6].

Il est indispensable de proposer rapidement une méthode de développement parfaitement adaptée aux L4G, sans quoi le développement "anarchique" qu'ils permettent posera, à terme, les mêmes problèmes de maintenance que ceux dans lesquels l'informatique de troisième génération s'est enlisée.

### *Problèmes quant à la philosophie d'utilisation*

Certains auteurs, trop enthousiastes quant aux possibilités des L4G, ou trop influencés par les promesses des fournisseurs de soft, envisagent carrément une informatique sans programmeurs. Ils se basent sur des cas réels, comme celui de la Santa-Fé Railroad, où les utilisateurs finals ont développé seuls une bonne partie de l'informatique opérationnelle de la firme à l'aide du L4G Mapper. L'expérience prouve qu'il s'agit là de cas exceptionnels, difficilement reproductibles. Il serait dangereux de croire qu'on peut confier la totalité du développement opérationnel à des utilisateurs: la pagaille ne tarderait pas. En

outre, on se permettra de douter du bon sens d'une proposition qui consiste à dire: "les informaticiens sont débordés ? Alors, que tout le monde devienne informaticien !" Ce serait oublier que les utilisateurs finals ont une tâche qui leur est propre, pour laquelle ils sont payés. Que penser d'une entreprise où tous les cadres doivent passer les deux tiers de leur temps à développer des applications informatiques ?

Cette idée fausse révèle un des écueils à éviter lorsqu'on utilise les L4G : ils permettent en effet de croire qu'on peut développer une application en se passant d'une bonne analyse. En L3G, la nécessité de l'analyse se fait ressentir très tôt: un programme non structuré devient vite illisible ! Les L4G, au contraire, par la guidance et la structuration automatique qu'ils offrent, permettent à l'utilisateur de s'avancer très loin dans le développement d'une application avant qu'il se rende compte qu'un problème se pose, qui fait parfois s'écrouler tout l'édifice. Les L4G facilitent le développement et la maintenance des applications, mais ne les rendent pas triviaux pour autant ! C'est pourquoi le développement d'applications complexes avec les L4G ne peut s'envisager que dans les structures classiques de l'informatique professionnelle.

En revanche, la tâche des utilisateurs finals peut être grandement facilitée par des outils informatiques faciles d'emploi, qui leur permettraient de répondre eux-mêmes à leurs besoins décisionnels, sans avoir à attendre le bon vouloir du CTI. Pour les aider à utiliser ces outils tout en respectant les contraintes de performance du système et de sécurité des données, on veillera toutefois à les encadrer, à leur offrir une structure d'assistance: l'infocentre.

### *Problèmes de performances*

On reproche souvent aux L4G leur performances médiocres: *"plus la sémantique est puissante et condensée, plus la navigation (laissée à charge de l'outil) dans l'ensemble des ressources disponibles risque de s'exercer en dehors du chemin optimal"* [BOUT.84]

Une étude récente prouve toutefois que la réputation de lenteur des L4G n'est plus tout à fait de mise [JALI.89].

En outre, l'utilisateur final, cible privilégiée des L4G, est susceptible de manipuler les outils qui lui sont offerts de façon inadéquate et peu performante (faire des sélections sans index,...).

## 2.2. Les méthodes de comparaison de langages

Pour choisir un outil adapté, il faut d'abord analyser l'environnement dans lequel il sera utilisé, puis en déduire un ensemble de contraintes permettant de sélectionner l'outil le plus adapté.

L'autre approche possible est de partir, non pas des contraintes de l'environnement d'utilisation réel, mais d'une analyse du "domaine d'application privilégié", c'est-à-dire des possibilités, de chaque L4G. Cette étude théorique de leur champ d'application aurait produit un panorama complet, fort utile puisqu'il permet d'effectuer un premier tri rapide des outils.

Comme nous l'avons dit dans l'introduction, nous avons préféré la première démarche, plus pratique car elle se base sur l'analyse des exigences des utilisateurs. Cette approche nous semble plus utilisable concrètement: elle permet de ne s'intéresser qu'aux problèmes réels de l'utilisateur, plutôt que de s'embarquer dans une étude théorique fastidieuse si on veut la mener avec rigueur et objectivité. Elle sera également plus fiable dans la pratique car elle fait appel à des tests adaptés au cas particulier plutôt qu'à des comparaisons théoriques générales, toujours floues lorsqu'il s'agit de les mettre en pratique. Ces études nous seront toutefois utiles pour "cibler" rapidement et à peu de frais le type de L4G qui convient, et pour se faire une première opinion sur chaque L4G sans avoir à le tester.

Une comparaison approfondie, basée sur le développement d'applications-test, est par ailleurs indispensable lorsqu'il s'agit de départager les outils qui semblent les mieux adaptés, mais son coût interdit qu'on l'applique à plus de deux ou trois L4G. D'où l'intérêt d'une pré-sélection rapide.

La littérature distingue principalement **deux méthodes de comparaison**:

- **l'approche expérimentale** préconise de développer quelques applications pilotes dans chaque langage, afin de déterminer lequel répond le mieux aux besoins. Toute la difficulté de cette méthode réside dans le choix des applications pilotes, qui doivent résumer de façon satisfaisante les contraintes de l'environnement.

- **l'approche par fonction** qui analyse la façon dont chaque langage aborde les groupes de fonctionnalités classiques telles que structures de contrôle, de données, ...

D'après [FEGE.84], seule l'approche expérimentale est adaptée lorsqu'on veut comparer des langages de type trop différent (procédural, applicatif, non-procédural,...). Etant donné la diversité des formes que peuvent prendre les L4G, notamment dans leur dosage de composantes procédurales / non-procédurales, on en déduit que l'approche expérimentale leur convient mieux. Toutefois, étant donné les coûts qu'entraîne une telle comparaison, il convient de la réserver aux deux ou trois L4G qui semblent à première vue les plus adaptés. C'est pourquoi l'approche par fonction, moins coûteuse à condition disposer de panoramas et essais pré-établis, nous semble indispensable pour effectuer un premier tri. Bien sûr, il n'est pas question de devoir établir au moment de la sélection un classement objectif et général des L4G disponibles: l'objectivité et la généralité coûtent cher en temps d'étude, d'autant plus qu'elles sortent du cadre de notre sélection qui se veut concrète, basée sur les contraintes d'un environnement bien précis. Ce travail théorique de fond est disponible dans de nombreux ouvrages [BOUT.84, MART.85, 86 a et b, ROUX.88, SVM, BYTE,...].

On distingue en outre deux façons de **répartir les tâches** de comparaison [FEGE.84] :

- **l'approche "advocatory"**, où l'on forme un spécialiste par langage; cet avocat défendra son langage dans tous les domaines de la comparaison, c'est-à-dire pour tous les groupes de fonctionnalités importants (structures de contrôle, données...).

- **l'approche coopérative**, où chaque groupe de fonctionnalités possède son spécialiste qui analysera chaque langage sous cet angle particulier.

Dans le premier cas, on demande un spécialiste par langage, dans le second, un spécialiste par catégorie de fonction.

Dans notre cas particulier, c'est-à-dire une sélection sur base du développement d'applications de test, l'approche "advocatory" demande de former un spécialiste par L4G afin de lui faire développer toutes les applications-test avec ce langage.

L'approche coopérative, au contraire, demande que chaque testeur se spécialise dans une application mettant en évidence une fonctionnalité particulière (générateur de rapports, ...), afin de développer cette application dans tous les L4G à tester.

On devine aisément les inconvénients de cette méthode coopérative: chaque testeur devra connaître chaque langage, du moins pour la partie ayant trait à la fonctionnalité qu'il étudie. Ce qui entraîne des risques de confusion importants entre les syntaxes des différents langages. Autre inconvénient: chaque L4G ne sera vu qu'en partie; personne ne pourra dire si ces parties forment un tout cohérent. En revanche, cette méthode offre l'avantage de gommer les différences d'aptitude entre testeurs, chacun d'entre eux pouvant, sans biais, classer les L4G selon les critères de leur propre domaine d'étude.

La méthode "advocatory", au contraire, ne permet pas de tirer de conclusions fiables quant aux efforts de formation et de développement requis par chaque L4G; il suffit en effet qu'un testeur soit plus lent à apprendre, ou plus mauvais

programmeur que les autres pour que le L4G qu'il est chargé de défendre soit pénalisé. Cette méthode permet toutefois d'analyser le langage dans son ensemble et d'en saisir rapidement la philosophie générale, ce qui rend le testeur capable de "plaider" en sa faveur ou, au contraire, de l'éliminer très vite s'il se révèle inadapté au cahier des charges.

Ces considérations nous permettent de croiser ces deux distinctions (par fonction/expérimentale et advocatory/coopérative) afin de proposer un **schéma global de comparaison**:

-1- la pré-sélection se fera par fonction et de manière "advocatory". Ces deux approches se combinent bien lorsqu'il s'agit d'évaluer rapidement et de façon globale l'adéquation d'un L4G au cahier des charges. La réunion de pré-sélection ne doit retenir que les deux ou trois meilleurs L4G parmi l'offre foisonnante du marché (parfois 15-20 L4G dans certains environnements). Elle ne peut s'embarrasser d'une comparaison trop nuancée, se noyant dans le détail des fonctionnalités de chaque langage, sans pouvoir tirer rapidement une conclusion d'ensemble. Au contraire, elle exige une évaluation globale, moins formelle et plus intuitive, donc plus rapide, ce qui correspond tout-à-fait à l'idée d'une réunion rassemblant les avocats de chaque langage, où chaque "plaidoirie" se baserait sur une évaluation de la philosophie du L4G à partir des fonctions qu'il offre (ou n'offre pas), et de leur intérêt pour le cas étudié.

Il est assez fréquent, lors de l'étape de pré-sélection, que l'on demande à chaque fournisseur de se faire l'avocat de son L4G, et de venir le défendre devant les "jurés" de l'entreprise. Les fournisseurs répondront à des questions basées sur les critères de sélection [cfr chapitre 4] de l'entreprise, ce qui évitera aux testeurs de perdre leur temps dans les documentations de L4G parfois totalement inadaptés. On peut également demander l'avis d'entreprises utilisant déjà les L4G analysés [GCM.86]. Le chapitre 4 analyse plus en détail les moyens disponibles pour "mesurer" les L4G.

-2- la sélection finale, elle, ne peut plus se contenter d'une évaluation floue. Les deux ou trois L4G restant en compétition se serrent de près; pour

les départager, des résultats concrets basés sur des tests sont indispensables (d'où une approche expérimentale). Ces tests devront comporter le moins de biais possible; pour ce faire, les temps de formation et de développement liés à une application-test (couvrant une fonctionnalité précise) devront être mesurés sur la même personne. D'où une approche coopérative: un spécialiste par fonctionnalité, chargé de l'évaluer dans tous les L4G restant en lice. Pour corriger les défauts attribués à cette méthode, à savoir la confusion des syntaxes et le manque de vue d'ensemble, on adoptera la démarche suivante: Après avoir développé leur application-test dans un langage, les spécialistes se regrouperont pour les intégrer, et juger des capacités du L4G à cet égard. Le cloisonnement des langages ainsi opéré contribuera aussi à diminuer les risques de confusion de syntaxe.

Dans certaines méthodes de sélection, il est fait appel aux fournisseurs pour développer eux-mêmes les applications-test. Malgré le gain de temps [la sélection peut se faire en deux jours ! GCM.86] et la moindre immobilisation de personnel que permet cette technique, elle nous semble peu intéressante car elle ne permet pas de tirer de conclusions fiables quant à l'effort de formation, à la facilité d'emploi,... Dans le cas cité [GCM.86], un seul fournisseur était parvenu à terminer les applications-test dans les délais impartis. Devait-on en conclure que le L4G "gagnant" était le meilleur, ou simplement que son fournisseur avait envoyé de meilleurs programmeurs ?

En n'utilisant que du personnel interne lors de la sélection finale, on peut au contraire juger "de l'intérieur" des capacités des L4G, de leur facilité d'emploi, d'apprentissage, ... C'est ce que préconise Martin lorsqu'il parle du "two-day test" [Mart.86.a]; peut-on être à l'aise avec le L4G et s'en servir utilement après une prise en main de deux jours? En outre, on disposera ainsi d'une équipe formée au L4G sélectionné lorsqu'il s'agira de le répandre dans l'entreprise.

Il faut être réaliste: le coût de la méthode de sélection d'un L4G doit être proportionnel au coût du L4G lui-même et aux bénéfices qu'on compte en retirer. Les modalités de déroulement de la sélection seront fixées par le schéma directeur informatique (voir chapitre 3): dépenses autorisées,

composition du comité d'évaluation (généralement un mélange d'informaticiens, d'utilisateurs et de hauts responsables), délais imposés,...

La méthode proposée ici est modulable quant à son coût; on peut l'appliquer de façon stricte et détaillée, en immobilisant plusieurs personnes pendant quelques semaines, s'il est question de choisir un L4G lourd destiné à devenir le langage standard de développement de l'entreprise. Par exemple, une entreprise peut décider que tout développement futur d'application se fasse en L4G, à l'exclusion des L3G utilisés avant.

On peut aussi appliquer cette méthode de façon plus grossière et moins coûteuse lorsqu'il s'agit de choisir un petit L4G très spécifique, destiné à un petit nombre d'utilisateurs finals. On pourra dans ce cas n'immobiliser qu'un seul testeur, voire même se contenter de la seule étape de pré-sélection.
Deux facteurs permettent de jouer sur le coût de la méthode: le nombre de testeurs et la durée des tests (déterminée par le nombre et la complexité de ces tests).

Ce qui nous permet de proposer le **plan de ce mémoire**:

## 1. Introduction

## 2. Etat de l'art:

- présentation des L4G: utilité, typologie, lacunes.
- présentation des différentes méthodes utilisées pour comparer les langages

# PARTIE II: METHODOLOGIE

## 3. Modélisation du cadre de décision

Répertorier, en toute généralité, les paramètres de l'environnement ayant une influence sur le choix de l'outil.
On aboutira à un schéma générique des contraintes, quel que soit le domaine d'utilisation.

## 4. Critères de sélection:

Les contraintes de l'environnement permettent de déduire les fonctionnalités attendues de l'outil, qui serviront de critère de sélection.
En outre, on tiendra compte de critères plus généraux, indépendants du contexte, qu'un "bon" L4G doit également respecter.
On obtiendra un schéma générique des critères, répondant à celui des contraintes, quelque soit le domaine d'utilisation.

## 5. Sélection:

*-1- Instanciation*

Instancier les paramètres de l'environnement; c'est à dire adapter au cas particulier le système général de contraintes et de critères de sélection des chapitres 3 et 4.

*-2- Pré-sélection*

Cette étape a pour but d'effectuer, à peu de frais, un premier tri des L4G.

-a- On déduira du système de contraintes quelle famille de L4G convient le mieux. Il existe des panoramas [BOUT.84, MART.86.a et b, comparatifs dans revues spécialisées] permettant de déterminer quels outils sont disponibles dans cette famille.

-b- Certains critères peuvent être évalués à peu de frais, en s'aidant de la documentation du L4G, de certains bancs d'essai publiés, de questions aux fournisseurs, de l'avis d'entreprises utilisant déjà un L4G..

*-3- Sélection*

Il s'agit ensuite de départager les deux ou trois L4G restant en lice en utilisant des critères plus fins qui ne peuvent être évalués sans un "banc d'essai". Il faudra donc mener des tests approfondis et mesurer la qualité de la réponse de chaque L4G à ces critères. Ce sera l'étape de sélection proprement dite.

# PARTIE II: EXPERIMENTATION

Cette expérimentation n'a pas pour but d'appliquer complètement la méthode proposée à un cas précis, mais d'étayer les affirmations de la partie théorique du mémoire, qui était uniquement basée sur la littérature.

On tentera de confirmer les liens établis entre les grands critères du chapitre 4 (non-procéduralité, intégration, convivialité,...) et les principales contraintes du chapitre 3 (maintenance, productivité, performances, flexibilité,...).

# PARTIE I : METHODOLOGIE

# 3. MODELISATION DU CADRE DE DECISION

D'après la méthode que nous avons choisie, la première étape dans le processus de sélection consiste à définir clairement ce qu'on attend du L4G à acquérir. Pour ce faire, nous allons tenter de relever, en toute généralité, les différentes contraintes à prendre en compte lors du choix d'un L4G, de quelque type qu'il soit, et indépendamment de tout cadre d'utilisation particulier.

En pratique, pour choisir l'outil le plus adapté à un cas précis d'utilisation, il faudra instancier ce schéma général des contraintes, c'est-à-dire pondérer les différentes contraintes en fonction de leur importance pour l'environnement d'utilisation étudié. Cette instanciation aboutira à un cahier des charges de l'outil.

On a montré qu'un L4G, surtout s'il est destiné à l'utilisateur final, doit être "taillé sur mesure" pour l'utilisation qu'on veut en faire, et donc qu'il serait utopique de vouloir satisfaire tous les besoins des utilisateurs de l'entreprise avec un seul L4G. Cela ne signifie pas pour autant qu'il faille acquérir un L4G par utilisateur ! Il faut plutôt raisonner en termes de type d'utilisateur, de tâche, d'environnement d'utilisation. D'où l'intérêt que représente une typologie des différents cadres d'utilisation possible d'un L4G: elle permettra de voir, dès le début, s'il est possible de trouver un L4G qui réponde à toutes les attentes, ou si l'entreprise doit, au contraire, s'équiper d'une gamme de L4G correspondant à sa gamme d'utilisateurs et de besoins. On choisira de préférence des L4G compatibles, pouvant échanger leurs données.

Le surcoût que représente l'achat de deux ou trois L4G pour utilisateurs finals, au lieu d'un seul, sera vite amorti par le désengorgement du CTI (centre de traitement informatique) qu'il permettra. De toutes façons, vu l'état de saturation auquel certains CTI sont arrivés, ont-ils encore vraiment le choix ? Devant la lenteur de réaction du CTI face à leurs besoins, les utilisateurs prennent souvent le taureau par les cornes et s'équipent eux-mêmes de L4G sur PC. Dans beaucoup de cas, l'anarchie s'installe; elle est due à une mauvaise

répartition des tâches entre un CTI débordé et des utilisateurs impatients. Un infocentre (voir chapitre 2) aura précisément pour rôle de satisfaire les besoins des utilisateurs tout en évitant cette anarchie.

Les L4G orientés développeurs, quant à eux, peuvent prétendre à un usage plus général, indépendant des contextes où tourneront les applications qu'ils vont aider à développer. Il serait d'autre part peu réaliste de vouloir équiper une entreprise de plusieurs L4G lourds, eu égard aux coûts d'acquisition, de formation, d'harmonisation que cela entraînerait.

En général, les entreprises optent donc pour un seul L4G lourd, et une gamme de L4G pour utilisateurs finals.

La constitution d'une gamme d'outils cohérente pour une entreprise ne peut se faire sans une analyse rigoureuse de ses besoins futurs en informatique, ainsi qu'une redéfinition des rôles respectifs du CTI et des utilisateurs quant aux tâches de développement d'applications. L'importance de ces choix pour l'entreprise impose qu'on les prenne en charge à un très haut niveau de responsabilité. La direction nomme souvent un comité directeur chargé de planifier la gestion informatique de l'entreprise; il consignera ce plan dans le schéma directeur informatique.

## 3.1. Le schéma directeur informatique

Le schéma directeur est le plan de développement de l'informatique dans l'entreprise. Il fixe les objectifs, les actions à mener, les moyens pour y arriver, ainsi qu'un calendrier de réalisation. Son horizon va de 3 à 10 ans (pour les schémas directeurs stratégiques). Le schéma directeur peut remplir plusieurs missions:

- informer et diagnostiquer l'existant

- formuler des scénarios d'évolution possible

- coordonner et programmer les actions

- en fixer l'organisation budgétaire

- contrôler et suivre leur application

- permettre le dialogue entre les différents acteurs informatiques

Le comité directeur chargé d'élaborer ce schéma sera composé de représentants de la direction, ainsi que de responsables de l'informatique et des services utilisateurs.

Il est important que la direction hiérarchique s'engage explicitement lors de tout investissement lourd, et un L4G en fera souvent partie. Sa vision à long terme apportera à ces choix la cohérence, la continuité et la sécurité nécessaires.

Le schéma directeur spécifiera le portefeuille d'applications dont l'entreprise a besoin, ainsi que leur niveau de priorité. Il décidera également si l'approche L4G doit être tentée, et quelles applications du portefeuille elle concernera. Enfin, le comité directeur nommera un comité d'évaluation chargé de choisir le ou les L4G à acquérir [BODA.90].

## 3.2. Les contraintes

Si l'on veut analyser les contraintes dont il faut tenir compte lors du choix d'un L4G, une approche basée sur les différents environnements auxquels ils seront confrontés dans l'organisation semble plus indiquée qu'une approche basée sur un cycle de vie de plus en plus discuté dans son application aux L4G. On a donc préféré une approche organisationnelle.

Les L4G agissent dans plusieurs contextes différents (utilisateurs finals et développeurs); ils s'adaptent mal au cadre trop strict du cycle de vie classique d'un projet informatique [cfr chapitre 2 et LAUR.89]. Une analyse basée sur le cycle de vie aurait mis en évidence les contraintes liées à un environnement de développement classique, souvent fort différentes de celles auxquelles sont soumis ces L4G. Leur facilité d'utilisation leur permet en effet de dépasser ce cycle de vie trop rigide (utilisateurs finals acteurs du développement, prototypage, réduction de la phase de conception, facilité de maintenance...), rendant caduques certaines contraintes classiques (par exemple, la modularisation lors de l'étape de conception est de plus en plus discutée).

Tant les L4G pour utilisateurs finals que les L4G orientés développeurs doivent tenir compte des contraintes que leur impose leur environnement.

Pour améliorer leur productivité, on les plonge directement dans l'environnement d'utilisation; on veut qu'ils soient opérationnels le plus vite possible. Cela les rend fort dépendants de cet environnement, et leur impose plus de contraintes qu'à un langage classique, sans cesse adapté par le CTI, qui perd d'ailleurs beaucoup de temps dans cette tâche.
Par exemple, le fait d'être utilisés par des non-informaticiens, ainsi que leur objectif majeur de soulager le CTI, leur impose de disposer, de façon standard et sans devoir être adaptés par le CTI, de modules de communication avec l'environnement dans lequel ils sont plongés. Ces langages doivent en effet pouvoir communiquer avec des applications et bases de données existantes, qui sont souvent leur principale source d'approvisionnement en données. Ces problèmes d'interconnexion sont beaucoup moins cruciaux pour les L3G et les L4G orientés développeurs, dans la mesure où un module de communication peut facilement être développé par les informaticiens du CTI; on ne peut demander à l'utilisateur final d'en faire autant pour son L4G.

Les L4G pour utilisateurs finals, étant directement plongés dans l'environnement d'utilisation, en subiront les contraintes de plein fouet.

Les L4G orientés développeurs, d'usage plus général, et toujours protégés par la tour d'ivoire du CTI, ne subiront ces contraintes qu'indirectement, via les spécifications des applications qu'on leur demande. Néanmoins, ils seront plus

dépendants d'autres contraintes telles que la coopération avec les applications ou progiciels existant déjà, la méthode de développement en vigueur au CTI, les ressources disponibles, ...

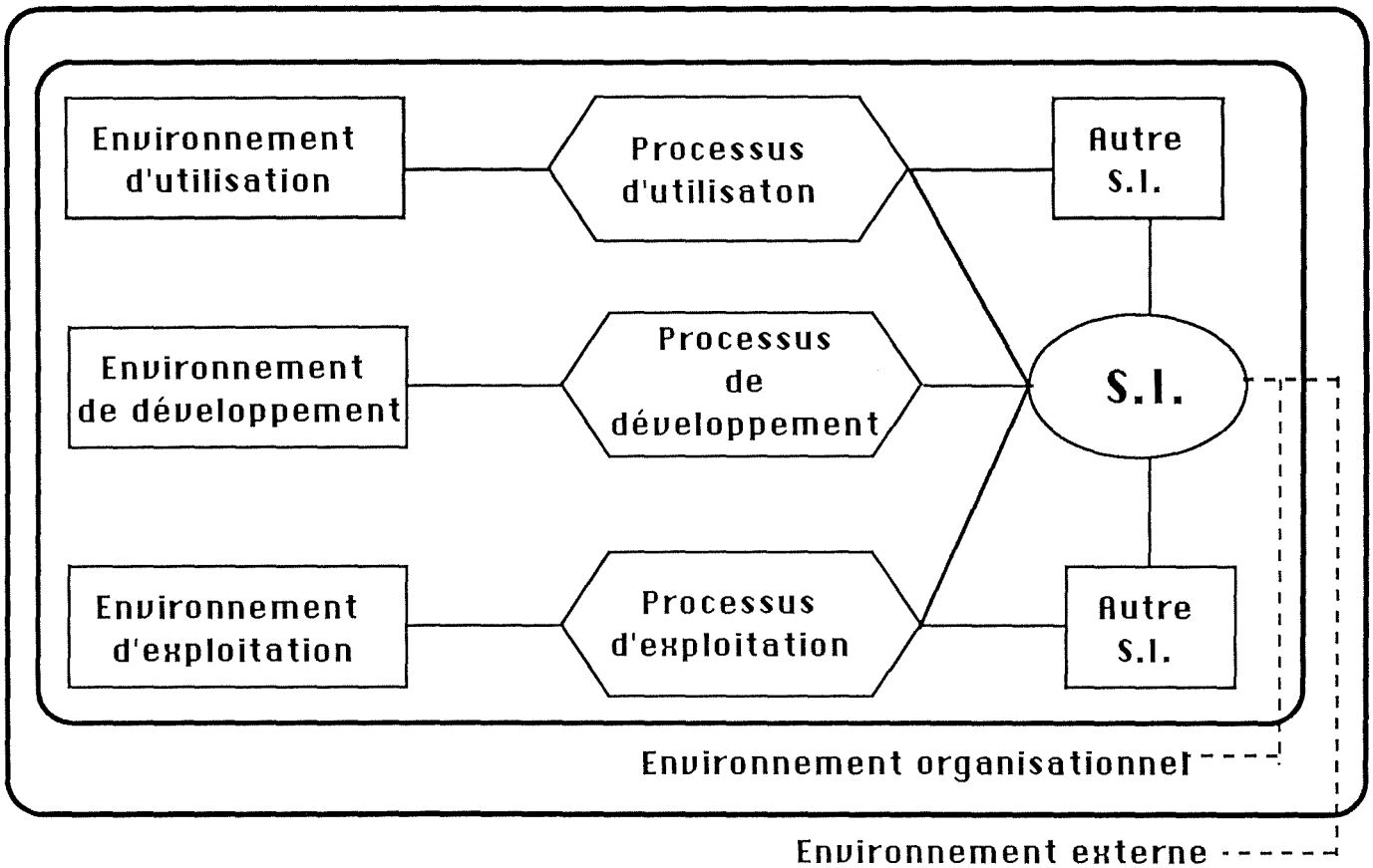
Le choix d'un L4G est à replacer dans le cadre plus large du schéma directeur informatique. C'est ce schéma qui impose une bonne partie des contraintes, et qui fixe beaucoup de paramètres dont la décision devra tenir compte, sans pouvoir rien y changer. Certains de ces paramètres ont pourtant une influence cruciale sur l'implantation du L4G; par exemple, la résistance ou la mauvaise formation des utilisateurs, les buts conflictuels d'un SI. C'est pourquoi nous écarterons de la procédure de choix toutes les contraintes dont la solution nous semble relever plus d'une décision du comité directeur que du comité chargé de choisir le L4G. Par exemple, la décision de sensibiliser les utilisateurs, de les former, de modifier la répartition de la tâche de développement d'applications, ...

Il ne sera jamais question, lors de la procédure de sélection, de modifier les décisions du schéma directeur. Il s'agira seulement de déterminer s'il existe un outil permettant de les mettre en oeuvre dans la réalité de l'entreprise. Cependant, si, de par l'incompatibilité de certaines contraintes, aucun outil ne semble convenir, on pourra avertir le comité directeur des risques que peut comporter une implantation du L4G sans modification du schéma. Par exemple, il serait hasardeux de vouloir implanter un L4G chez des utilisateurs finals résistants, surtout si on en attend beaucoup de leur part ... La méthode proposée ici permet donc de considérer l'ensemble des contraintes pouvant influencer la sélection, afin de pallier aux négligences ou imprécisions dues à la trop grande généralité du schéma directeur.

Les contraintes ayant une incidence directe sur les caractéristiques attendues du L4G [cfr critères du chapitre 4] seront regroupées, pour chaque environnement, dans la rubrique "contraintes influençant le choix".
Les contraintes plus générales, ayant une influence, non sur le choix du L4G, mais sur le succès de son implémentation, seront également relevées, dans la rubrique "contraintes générales".

Pour représenter ces environnements, nous avons choisi le modèle de Ives [LESU.89.a]. Son but initial était d'identifier et de classer les problèmes économiques, sociaux, politiques et techniques posés par les systèmes d'information.

## *Modèle de Ives:*




Chaque environnement représente un réservoir de ressources et de contraintes pour le processus correspondant. Les processus représentent les interactions entre le SI en ses environnements. Dans le cadre qui nous intéresse, l'aspect explicatif de ce modèle sera délaissé; on s'en servira seulement pour répertorier et classer les contraintes subies par un L4G. Puisque nous ne nous intéressons qu'aux contraintes, seuls les environnements, tels que définis par Ives, nous intéresseront.

Ives distingue cinq environnements, auxquels il convient d'en ajouter deux: celui des autres SI existant dans l'entreprise (autres applications, progiciels, bases de données), et celui du SI qu'on se propose de développer par L4G.

Nous tenterons de relever et de classer les contraintes qui naissent de chaque environnement.

Comme on l'a fait remarquer plus haut, certains environnements concernent plus les L4G pour utilisateurs finals, d'autres les L4G pour développeurs; nous le signalerons en cours de route.

## 3.3. L'environnement externe

### *1. Les contraintes légales, sociales et politiques externes*

Citées par Ives, elles ont une influence lointaine et difficile à évaluer sur le choix d'un L4G. Nous ne les retiendrons pas.

### *2. Les contraintes technologiques*

L'offre en matière de ressources informatiques, ont une grande influence sur l'évolution des langages informatiques. Certaines nouveautés semblent en effet avoir contribué au succès du phénomène "quatrième génération":

*** offre hardware:**

- l'apparition des PC, stations de travail, et même des mini-ordinateurs départementaux ou dédiés à la gestion de BD (IBM S/38), bref, la diffusion de la puissance de calcul

- de même, la convivialité croissante des micro-ordinateurs (écrans graphiques,souris...), devenant la référence en matière d'ergonomie

*** offre software:**

- l'apparition des bases de données relationnelles

- les tableurs

- les interfaces graphiques à "manipulation directe" (via la souris et les objets de l'interface)

- l'intégration croissante d'outils jusque là bien séparés (SGBD, calcul, graphiques, générateurs d'écrans, d'applications, langages procéduraux...)

Bref, la technologie de l'information a exercé une influence déterminante pour le succès des L4G. Nul doute qu'elle en exercera encore beaucoup sur leur évolution. D'où la nécessité d'en tenir compte lorsqu'il s'agit de choisir un L4G.

## *3. Le niveau moyen de formation à l'informatique*

Maturité informatique des différentes catégories d'employés de l'entreprise; cela permet d'évaluer l'effort de formation nécessaire, le niveau de convivialité de l'outil à acquérir,...

## *4. L'utilisation de l'informatique chez les concurrents*

Il s'agit souvent d'essayer de rattraper le retard pris sur eux, sous peine de n'être plus compétitif.

Les technologies de l'information ne sont plus seulement utilisées pour rationaliser la production et réduire les coûts, elles servent aussi d'arme compétitive permettant de se battre sur le marché. Elles peuvent même procurer à l'entreprise un avantage stratégique sérieux sur ses concurrents. [BODA.90]

Il est difficile d'évaluer l'impact des contraintes légales, sociales, politiques et concurrentielles sur le choix d'un L4G. On en tiendra compte compte lors de l'élaboration du schéma directeur informatique. Ce niveau d'analyse, lié à la stratégie de l'entreprise, dépasse le cadre de ce mémoire, qui ne s'intéresse qu'à une des décisions à prendre en exécution du schéma directeur: celle du choix d'un L4G.

## *5. Contraintes retenues:*

### -a- influençant le choix

- l'offre technologique (hard et soft): il faut suivre de près l'évolution de l'offre, et ne négliger aucune piste prometteuse.

- le niveau moyen des connaissances informatiques dans le marché de l'emploi, pour les différentes catégories de personnel, et son influence sur la facilité d'utilisation pour des non-informaticiens

**-b- générales**

- le niveau moyen des connaissances informatiques, pour son influence sur l'effort de formation à prévoir

## 3.4. L'environnement organisationnel

### *1. Les objectifs informatiques*

Les objectifs généraux de la firme se trouvent en général dans le schéma directeur informatique, s'il existe. On tiendra compte notamment de:

**-a-** la place que le L4G occupera dans le plan de développement d'applications; sera-t-il le seul langage de développement, ou, au contraire, sera-t-il entouré de L3G, de progiciels, ou d'autres L4G ? Devra-t-on opter pour un L4G lourd, d'usage général, capable de remplacer totalement le Cobol (orienté développeurs) ? Peut-on se contenter, au contraire, d'un outil plus léger, plus spécifique, d'usage moins universel (orienté utilisateurs finals) ?

**-b-** la place des utilisateurs finals dans l'utilisation du L4G: seront-ils cantonnés à leur rôle classique d'utilisateur peu consulté, ou joueront-ils un rôle plus actif avec un L4G convivial ? La transition entre ces deux rôles ne peut se faire sans une bonne formation; elle implique en outre un changement dans les habitudes et les mentalités. On tiendra donc compte également de la différence entre les rôles joués par les utilisateurs finals avant et après l'acquisition du L4G.

**-c-** la place du CTI (centre de traitement informatique) dans l'utilisation du L4G; sera-t-il maître d'oeuvre ou seulement consultant ?

Une perte de pouvoir du CTI dans tout ce qui touche à l'informatique ne se fait jamais sans grincements de dents...

**-d-** le but attribué au(x) SI à développer avec le L4G:

- est-il clair, précis, non-ambigu ?Des buts imprécis amèneront des spécifications floues.

- est-il conflictuel ?
(but de rationalisation, d'augmentation de la productivité, voire de compression de personnel). Des buts conflictuels risquent d'amener de la résistance chez les utilisateurs, ce qu'il faut éviter à tout prix puisqu'un L4G compte sur leur participation active.

- le SI s'intègre-t-il dans une stratégie commerciale de diversification des activités de la firme, de recherche d'avantages compétitifs via les ressources de l'information ? Ou vise-t-il seulement une rationalisation, un gain de productivité ? Dans le premier cas, on recherche avant tout une meilleure qualité des décisions (informatique décisionnelle, exploratoire). Dans le second, on recherche surtout la productivité, la rentabilité (informatique opérationnelle). A ces deux cas correspondent des L4G très différents.

Ces S.I. sont définis dans le portefeuille d'applications du schéma directeur.

**-e-** quelle priorité, quel délai accorde-t-on au développement de ce(s) SI ? S'il y a urgence, la rapidité de mise en oeuvre d'un outil sera préférée à d'autres critères tels que la polyvalence, la convivialité, ...

**-f-** quelle dépense tolère-t-on pour l'acquisition du L4G ?

## *2. La structure de l'entreprise*

Est-elle centralisée, ou favorise-t-elle la prise d'initiatives au sein de petites unités ? Des bureaucrates sous surveillance ne tireront pas autant d'avantages de leur L4G que des utilisateurs enthousiastes.

## *3. Le style de management*

- est-il participatif ou autoritaire ?

- néglige-t-il les facteurs humains ?(conditions de travail trop lourdes, stress,..)

## *4. La présence d'une structure d'assistance aux utilisateurs*

Un infocentre performant et bien accepté par les utilisateurs est un gage de réussite lorsqu'on implante un L4G pour utilisateurs finals. C'est une structure de formation et de soutien irremplaçable pour faire accepter un nouveau type d'outil.
En outre, sa fonction de standardisation permet d'éviter aux entreprises "laxistes" de voir l'anarchie s'installer dans les L4G utilisés par les utilisateurs finals (par exemple, il est préférable de n'utiliser qu'un seul tableur, afin de pouvoir échanger les données).

## *5. La résistance*

La motivation des utilisateurs est un facteur critique de succès dans tout projet d'informatisation. L'utilisateur devra être d'autant plus motivé qu'on attend beaucoup de lui. C'était déjà vrai pour les méthodes de développement traditionnelles, ça l'est plus encore pour les méthodes "participatives" souvent liées aux L4G. Si l'on attend de l'utilisateur final qu'il participe activement au

développement, voire qu'il développe lui-même, il faut qu'il prenne des initiatives, qu'il se soit approprié du projet. Le moindre indice de résistance de sa part devrait agir comme une sonnette d'alarme incitant à remettre en cause l'approche du projet, voire le projet lui-même (répartition des tâches, gain de productivité attendu, effort de sensibilisation, plan de formation, ...).

Il en va de même pour la résistance que pourrait montrer le CTI s'il perd son rôle de maître d'oeuvre pour passer à celui de simple consultant.

L'introduction d'un L4G peut bouleverser les habitudes de travail, la distribution des tâches, et donc la répartition des pouvoirs. D'autant plus que le L4G est parfois utilisé comme "*moteur de changement*" [HAIN.88]. Autant de facteurs générateurs de résistance si l'on n'y prend garde.

En tout état de cause, un L4G, aussi convivial soit-il, ne pourra rien contre cette résistance; il serait dangereux de croire qu'un L4G peut, à lui seul, résoudre ce problème. La solution est de niveau organisationnel; elle dépasse largement le simple choix du L4G. La résistance est donc une contrainte dont on tiendra compte à un niveau plus élevé que celui qui nous intéresse. Son incidence sur le choix d'un L4G se fera indirectement, via le comité directeur, qui seul pourra décider d'introduire le L4G plus progressivement, de chercher les causes de résistance, de mener une campagne de sensibilisation, de modifier la répartition des tâches, ...
En changeant l'orientation du projet, il dirigera automatiquement la décision vers un autre type de L4G.

## *6. Contraintes retenues:*

### -a- influençant le choix

- la place du L4G parmi les autres outils de développement d'applications. Veut-on un L4G pour les utilisateurs finals, ou pour les développeurs ? Doit-il remplacer le Cobol ? etc

- le rôle des utilisateurs finals dans l'utilisation du L4G, et donc le type d'outil: pour développeurs ou pour utilisateurs finals ?

- la priorité (le délai) accordée au développement du (ou des) SI avec le L4G.

- but de rentabilité (L4G opérationnel) ou de qualité de la décision (L4G décisionnel) ?

- buts flous, donc besoin d'outils d'aide à la spécification (prototypage)

- la dépense tolérée pour l'acquérir.

- la présence d'un infocentre et d'une tradition d'encadrement des utilisateurs

**-b- générales**

- le changement de rôle demandé à l'utilisateur final, et donc l'importance de l'effort de formation, de sensibilisation, les risques de résistance, ...

- la place du CTI dans l'utilisation du L4G, et les risque de résistance s'il perd du pouvoir

- les risques de résistance de la part des utilisateurs finals, provenant de buts conflictuels, d'une structure trop centralisée, d'un management autoritaire, qui tuent habituellement les initiatives, et doivent donc s'attendre au pire lorsqu'ils en demandent à leurs employés.

## 3.5. L'environnement d'utilisation

On s'intéresse ici à l'utilisateur final, celui dont le travail, les décisions, sont valorisées par l'informatique. C'est le véritable bénéficiaire du SI.
On le distingue de l'utilisateur apparent (personnel d'exploitation), qui n'est qu'un intermédiaire entre l'informatique et l'utilisateur final.
Cet environnement exerce ses contraintes de façon directe sur un L4G pour utilisateurs finals, et de façon indirecte sur un L4G pour développeurs, via les spécifications des applications qu'il doit aider à développer.

### *1. Le profil de l'utilisateur*

#### -a- sa formation

Un juriste et un ingénieur ont une attitude et une réceptivité différentes face à l'informatique. Un ingénieur sera plus ouvert aux nouvelles technologies, il sera moins réticent, moins craintif, plus motivé. Au risque de trop généraliser, nous pensons que la convivialité d'un outil aura pour lui moins de poids que d'autres critères tels que la puissance, les performances,..., au contraire d'un juriste ou autre "littéraire".
En outre, sa formation scientifique l'aura mieux préparé à la logique et aux raisonnements formels qu'implique toute tâche de programmation, si légère soit-elle.
La formation de base exercera donc de l'influence sur la formation et l'assistance à prodiguer, la productivité à espérer, et sur la convivialité demandée au L4G, seul critère qui nous intéresse directement.

### -b- sa fonction dans l'entreprise

L'expert et le décideur n'ont pas les mêmes besoins, la même utilisation, ni le même temps à consacrer à l'informatique. Le décideur a besoin d'outils de communication, de synthèse d'informations, et d'aide à la décision. Les opérations qu'il exécute sont très diversifiées et peu standardisées [LESU.89.b]. Il passe la moitié de son temps en réunion, où la cadence de travail est élevée; pour lui, l'informatique doit être rapide et efficace, sans devoir y consacrer trop de temps. Le décideur développera rarement lui-même ses applications, faute de temps; il devra plutôt spécifier ses besoins à un tiers, qui sera chargé de préparer la décision (agrégation d'informations, présentations synthétiques,...).

L'expert, lui, travaille surtout dans son bureau, pour préparer les décisions des managers. Il passe moins de temps en réunion, sa cadence de travail est moins élevée. Ses tâches sont moins variées, il manipule des informations plus formalisées et plus détaillées, en utilisant des modèles [LESU.89.b].

La fonction de l'utilisateur dans l'entreprise aura donc une grande importance sur le choix du L4G quant à ses fonctionnalités, à sa convivialité, et à l'effort de formation nécessaire. La fréquence avec laquelle l'utilisateur se sert de l'outil aura également beaucoup d'importance: a-t-on affaire à un novice qu'il faut rassurer et guider, à un utilisateur intermittent auquel il faudra rafraîchir la mémoire, ou à un utilisateur expérimenté qui souhaite travailler le plus vite possible?

## *2. La place de l'utilisateur dans le circuit de développement*

L'utilisateur final peut exercer trois rôles dans le processus d'informatisation:

- utilisateur simple, son rôle traditionnel, seulement consulté pour la spécification de ses besoins.

- analyste plus ou moins actif auprès des informaticiens professionnels pour le développement de son application

- développeur à part entière de l'application couvrant ses besoins.

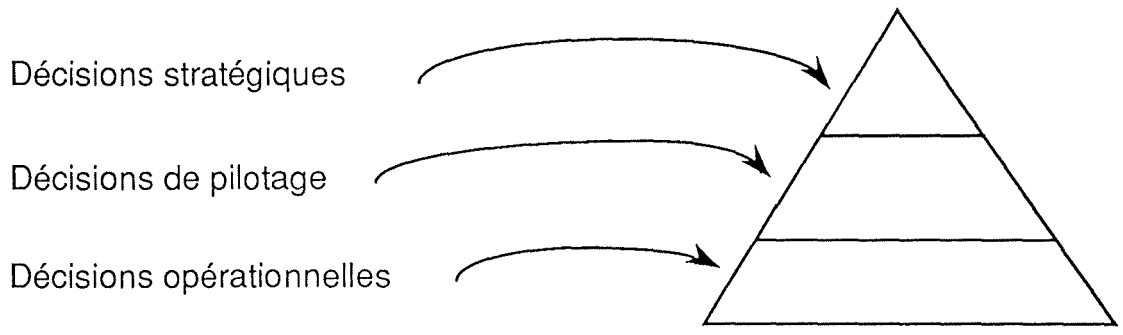
Sa place lui est attribuée par le comité directeur, qui veillera aussi à établir un plan de formation si ce rôle est nouveau. Un rôle actif impose de pouvoir consacrer du temps à la formation à l'outil, ainsi qu'à une utilisation complète de ses possibilités (y compris l'analyse et le développement, si besoin est). Cela impose en outre de s'orienter vers un L4G pour utilisateurs finals.

## *3. Le type de décision*

On classe traditionnellement les tâches et les décisions qui s'y rapportent suivant deux axes [BODA.89.a, LESU.89.a]: leur niveau et leur degré de structuration.

### -a- niveau de décision

On distingue trois niveaux de décision:



Cette distinction est basée principalement sur le niveau hiérarchique qui prend la décision. Toutefois, cette découpe en niveaux peut se justifier selon d'autres critères:

| Critère | opérationnel | pilotage | stratégique |
|---|---|---|---|
| source renseignements | interne | ---------- | externe |
| niveau d'exactitude | précis | | ordre de grandeur |
| niveau de détail | fin | | synthétique |
| horizon décision | court terme | | long terme |
| fréquence décision | quotidienne | | rare |
| âge décision | récente | | ancienne |

L'objectif poursuivi par l'informatique est différent selon qu'il s'agit d'aider la prise de décision à tel ou tel niveau. Globalement, l'informatique opérationnelle visera une amélioration de la productivité, de l'efficacité des tâches et décisions opérationnelles. Alors que l'informatique stratégique (décisionnelle) visera à faciliter la découverte d'opportunités, de menaces, ... bref, à aider la prise de décision, voire à améliorer sa qualité.

### -b- degré de structuration

Le second axe de classement des décisions est leur degré de structuration:

- **structurées**, quasi-algorithmiques; l'informatique peut jouer un rôle central, l'homme peut se limiter au contrôle du bon déroulement

- **semi-structurées**, où l'algorithme doit s'accompagner de jugement humains (ou de systèmes experts)

- **informelles**, où le flair, l'expérience et l'heuristique prédominent; l'homme a un rôle central, l'informatique un rôle d'appoint (logistique, synthèse et présentation d'informations ...)

Une décision structurée peut être facilement spécifiée, planifiée; elle peut prétendre à une plus grande stabilité dans le temps. C'est le domaine privilégié de l'informatique lourde traditionnelle, visant à remplacer l'homme là où il n'est plus nécessaire.
Une décision informelle ne peut se reposer sur un modèle théorique; ses spécifications sont floues, changeantes. C'est le domaine de l'exploration et du "*ad hoc computing*" [MART.85].

Si l'on croise les deux axes (par niveau et par structuration), on obtient le tableau suivant, accompagné d'exemples [BODA.89.a, LESU.89.a] :

| **Critère** / **Méthode** | *opérationnel* | *pilotage* | *stratégique* |
|---|---|---|---|
| *structurée* | gestion de stock | choix de lignes de crédit | localisation d'une usine |
| *semi-structurée* | Page de couverture du "Times" | budget, marketing | fusion ou OPA ? |
| *non-structurée* | | embauche de responsables | projet à long terme (R&D,...) |

Les décisions structurées et opérationnelles privilégient la productivité, l'efficacité, le contrôle des coûts; elles sont planifiées, pré-spécifiées, stables.

Les décisions non-structurées et décisionnelles privilégient la qualité de la décision, la découverte d'opportunités, l'interactivité, le "brain-storming"; elles sont imprévisibles, changeantes, floues et difficiles à spécifier.

## *4. Le domaine d'application*

La distinction entre les domaines d'application de l'informatique est vraie pour les L4G autant que pour les L3G. Les deux grands pôles restent l'informatique scientifique, gourmande en calcul, et l'informatique de gestion, traitant de gros volumes de données.

## *5. Contraintes retenues*

### -a- influençant le choix

- la formation de base de l'utilisateur final, et donc la convivialité attendue de l'outil.

- sa fonction dans l'entreprise, et donc le type de données manipulées, ainsi que les fonctionnalités, la convivialité, la durée d'apprentissage, la façon de se servir du L4G.

- la fréquence d'utilisation de l'outil.

- la place de l'utilisateur dans le circuit de développement, qui détermine la philosophie de l'outil (orienté développeurs ou utilisateurs finals).

- le niveau de décision:

opérationnelle-- but de performance; L4G opérationnel
stratégique------ but de synthèse d'informations; L4G décisionnel

- le degré de structuration de la décision :

structurée --- but de remplacer l'homme; stabilité
informelle --- but d'aider l'exploration; instabilité

- le domaine d'application: gestion ou scientifique

**-b- générales**

- la formation de base de l'utilisateur, et donc l'effort à consentir pour le former à l'outil et le lui faire accepter

- le changement de rôle qui lui est demandé, et donc l'effort de formation à prévoir, mais aussi les risques de résistance

- le rôle qu'on veut lui attribuer dans le développement est-il compatible avec sa fonction, notamment quant au temps qu'il peut consacrer à l'outil, et la façon dont il va s'en servir ?

## 3.6. L'environnement de développement

L'environnement de développement classique regroupe des informaticiens professionnels (analystes et programmeurs); il dépend généralement du centre de traitement informatique (CTI). Il sera important d'en tenir compte si l'on doit choisir un L4G orienté développeurs.

### *1. Les méthodes d'analyse*

Les méthodes utilisées classiquement sont adaptées aux applications lourdes, stables et planifiées du niveau opérationnel. Elles sont dominées par les experts, les analystes professionnels. Elles doivent par contre être modifiées

pour tenir compte d'applications moins stables, plus difficiles à spécifier, en faisant appel à une plus grande participation des utilisateurs.

*"Les L4G se rapprochent plus du domaine de la spécification que de celui de la réalisation. D'autre part, les rapports entre les composants BD, dialogues et traitements s'y trouvent bouleversés: les traitements y apparaissent comme subordonnés aux composants de la base de donnée et des dialogues."* Une méthode adaptée aux L4G est d'ailleurs proposée dans ce mémoire: [LAUR.89]

J. Martin, quant à lui, préconise un découpe des applications en modules individuels; il semblerait que certains L4G atteignent leur productivité maximale lorsqu'ils sont utilisés par une seule personne.

Si la méthode utilisée n'est pas adaptée aux L4G, et qu'on ne veut malgré tout pas en changer, on pourra s'orienter vers un L4G "traditionnel", ne bouleversant pas trop les habitudes de travail [cfr chapitre 2].

## *2. la place du CTI dans le circuit de développement*

Si les informaticiens du CTI perdent leur rôle de maître d'oeuvre dans le processus de développement, on peut s'attendre à des problèmes de résistance de leur part. Pourront-ils se contenter d'un rôle de consultant au service de l'utilisateur, ce qui n'est pas dans leurs habitudes ?

## *3. le degré de saturation du CTI*

Est-il à ce point élevé qu'on en arrive à choisir un outil ne visant que la productivité immédiate, ou préfèrera-t-on ménager l'avenir et opter pour un outil ne compromettant pas la maintenance future, ni la réutilisabilité des applications développées ? [voir chapitre 4]

Pourra-t-il libérer des informaticiens pour assister les utilisateurs ? Faudra-t-il au contraire envisager de recruter à l'extérieur les membres de l'infocentre ?

## *4. Contrainte retenues:*

### -a- influençant le choix

- les méthodes d'analyse utilisées

- la place du CTI dans le développement, déterminant le type de L4G (développeurs ou utilisateurs finals)

- le degré de saturation du CTI, pour déterminer si la productivité doit être à tout prix recherchée

### -b- générales

- le changement de rôle demandé au CTI, et donc les risques de résistance, la nécessité de former à ce nouveau rôle de consultant

- si on envisage d'opter pour une méthode d'analyse plus adaptée aux L4G, il faudra également former les informaticiens à cette nouvelle méthode.

- le degré de saturation du CTI lui permettra-t-il de jouer le rôle qu'on veut lui attribuer (assurer l'assistance, ...) ?

## 3.7. L'environnement opérationnel

Le matériel et l'environnement d'exploitation disponibles imposent deux types de contraintes sur le choix d'un L4G:

### *1. les ressources disponibles*

Les ressources disponibles, aussi bien humaines que hardware (CPU, mémoires de masse, ...), sont déterminantes lorsqu'on envisage de s'équiper d'un L4G. Ces langages sont en effet très gourmands, aussi bien en temps CPU qu'en place mémoire. Ils facilitent en outre l'accès aux données, rendant le maintien de leur intégrité difficile, et provoquent une inflation de la demande.
Les ressources disponibles pourront-elles supporter le surcroît de travail demandé par le L4G, tout en assurant des performances et une fiabilité acceptables ? Doit-on au contraire envisager un upgrade du matériel, la création d'un poste d'administrateur des données, d'un infocentre ... ?

### *2. la richesse de l'environnement*

L'environnement exploité dans le centre où doit être implanté le L4G détermine la gamme de langages envisageables. Certains environnements sont très fournis en L4G (MS-DOS, Macintosh, MVS), d'autres ont moins de chance.

*3. Contraintes retenues:*

**-a- influençant le choix**

- les ressources disponibles

- l'environnement d'exploitation

**-b- générales**

- est-il réaliste de vouloir utiliser un L4G sans upgrade du matériel, sans modification de la structure d'exploitation ?

- peut-on envisager de migrer vers un environnement plus riche en L4G, au cas où le nôtre serait trop pauvre ?

## 3.8. Le SI à développer par L4G

Cet environnement concerne aussi bien les L4G orientés développeurs que ceux destinés aux utilisateurs finals. Cependant, son influence est difficile à évaluer, pour des raisons différentes chez l'un et l'autre:

- il sera difficile d'obtenir des pré-spécifications précises de la part des utilisateurs finals.

- même si elles sont parfaitement spécifiées et planifiées, le nombre et la diversité des applications qu'est appelé à développer un L4G orienté développeurs rend toute évaluation globale de la tâche difficile.

Néanmoins, un ordre de grandeur sera souvent suffisant pour définir les applications-test sur lesquelles on départagera les L4G.

La découpe d'une application en ses composantes BD, traitements et dialogue semble beaucoup plus nette pour les L4G qu'elle ne l'était pour les L3G; il est dès lors préférable que cette découpe soit effectuée très tôt dans l'analyse fonctionnelle, afin de coller de près aux concepts de base des L4G [HAIN.88, LAUR.89].

Il est possible de repérer de quels objets de chaque type (BD, traitements, dialogues) l'application sera composée. Pour ce faire, l'idéal serait de disposer déjà de son analyse fonctionnelle complète; toutefois, une première évaluation des entités et des fonctionnalités devrait suffire.

Pour chaque type d'objet, on précisera comment mesurer leur taille, leur nombre et leur complexité, ainsi que le nombre et la complexité des relations qui les unissent. Cela permettra de de déterminer si les L4G étudiés supportent facilement ces volumes, cette complexité, ces relations.

Pour de plus amples informations concernant la modélisation des composantes des L4G, voir [LAUR.89], d'où est extraite la décomposition en objets sommairement présentée ici.

## *1. la Base de Données*

### -a- les objets

Basée sur le modèle relationnel, son objet central est la relation, ou table. Elle est composée d'un ensemble de champs possédant un type particulier (numérique, chaîne de caractères,...). Ces champs peuvent servir de clé d'accès (unique ou pas, indexée ou pas) aux records, ou occurrences, de la relation.

### -b- les relations entre objets

L'existence ou le contenu d'un objet peut être lié à l'existence ou au contenu d'autres objets. On appelle ces liens entre les objets "contraintes d'intégrité" (CI). La BD est dite cohérente si elle respecte l'ensemble des CI définies sur ses objets. Les principaux types de CI sont: la connectivité, l'identification, l'existence, l'inclusion, l'exclusion, l'égalité, le sous-typage, le domaine de valeurs et la dépendance fonctionnelle [voir BODA.89.a pour plus de détails].

### -c- paramètres déterminants

- *Complexité des objets*

La complexité d'une relation peut se mesurer au nombre de champs, de clés d'accès, d'index qu'elle possède.

- *Taille des objets*

La taille d'une relation équivaut à la somme des tailles de ses champs; la taille d'un champ est liée à son type: une chaîne de 50 caractères prend plus de place qu'un entier.

- *Nombre d'objets*

On distingue:

- le nombre de relations que comporte la BD

- le nombre d'occurrences (records) de chaque relation de la BD

- *Nombre de relations entre objets*

Le nombre de CI existant entre les objets.

- *Complexité des relations entre objets*

Le nombre d'objets différents impliqués dans la vérification d'une CI, ce qui correspond en fait à la complexité de la CI elle-même.

## *2. le dialogue*

### -a- les objets

Le dialogue avec un L4G se fait de trois façons: via des formulaires (écrans) ou des rapports (états imprimés), via des menus, et, classiquement, via un langage de commande.
Un formulaire/rapport (F/R) est un écran (ou état imprimé) comportant des blocs ou groupes (B/G) de champs provenant de diverses relations BD. Il pourra comporter, en outre, des boutons [voir point 4:pilotage]. Chaque B/G correspond à une "vue", à une donnée agrégée, composée d'une relation principale (dont les champs seront affichables et modifiables), et éventuellement de plusieurs relations secondaires liées à la principale par la correspondance de deux champs de même type (les champs de ces relations secondaires ne seront qu'affichables, via des variables, calculées ou non). *"Les B/G sont hiérarchisés au sein du F/R. Un B/G fils est lié à son B/G père par la mise en correspondance de deux champs de même type appartenant aux deux B/G liés... Cela signifie qu'un F/R permet indirectement de hiérarchiser des relations de la base de données pour constituer une vue sur une donnée agrégée et pouvoir effectuer des opérations sur cette donnée."* [LAUR.89]. Il faut noter enfin que des CI peuvent aussi être définies sur les champs d'un B/G; nous les avons déjà analysées plus haut. Les menus, quant à eux, sont des listes de commandes (items) permettant, comme les boutons, de déclencher des actions [voir point 4: pilotage].

### -b- les relations entre objets

Etablir des relations entre les objets du dialogue équivaut à spécifier la dynamique d'enchaînement des F/R, traitements, à hiérarchiser les événements; cela sera étudié dans le point 4:pilotage.

### -c- paramètres déterminants

- *Complexité des objets*

La complexité d'un F/R se mesure au nombre de B/G, et donc de relations BD, impliquées dans sa hiérarchie de B/G.

- *Taille des objets*

La taille d'un F/R correspond à la somme des tailles des B/G qui le composent; la taille d'un B/G correspond à la somme des tailles des champs de relations BD ou de variables qui le composent. La taille d'un champ ou d'une variable est liée à son type.
La taille d'un menu se mesure par le nombre d'items qui le composent, ainsi que par la profondeur de sa hiérarchie de sous-menus; un item peut en effet donner accès à un nouveau menu [voir point 4:pilotage].

- *Nombre d'objets*

Nombre de F/R et de menus.

## *3. les traitements*

### -a- les objets

Les traitements offerts par un L4G peuvent prendre des formes aussi différentes que des algorithmes, des modèles (tableurs), des règles d'inférence,... Les traitements peuvent être de deux types: procéduraux ou non-procéduraux (déclaratifs).

Les traitements procéduraux prennent la forme classique de procédures (ou formules, ou scripts,... selon la terminologie du L4G). Elles manipulent des relations BD, des champs de relations ou de B/G ou des variables. Elles peuvent en outre appeler d'autres procédures ou des F/R.

Les traitements non-procéduraux permettent, quant à eux, d'agréger un champ d'un F/R sur base d'autres champs (somme, moyenne, maximum,...), ou de spécifier le contenu d'un champ sur base d'une expression déclarative de calcul.

### -b- les relations entre objets

Les relations entre procédures sont constituées des appels qui sont faits de l'une à l'autre, ce qui correspond en fait à la dynamique de l'application [voir point 4: pilotage].

### -c- paramètres déterminants

- *Complexité des objets*

Grossièrement, la complexité des traitements procéduraux se mesure au nombre d'objets qu'il manipule (relations, B/G, champs, variables), au nombre d'objets qu'il appelle (F/R ou autre procédure), et enfin au nombre et au degré d'imbrication de ses structures itératives.
En ce qui concerne les traitements non-procéduraux, leur complexité est liée au nombre d'opérandes qu'ils utilisent.

- *Taille des objets*

La taille d'une procédure se mesure au nombre d'instructions qui la composent.
La taille d'une expression déclarative se mesure à la longueur de la formule qui la décrit.

- *Nombre d'objets*

Le nombre de procédures, d'expression déclaratives et d'opérations d'agrégation.

## *4. le pilotage*

Les L4G avancés disposent d'une fonction chargée de gérer et de contrôler l'enchaînement des actions: la fonction de pilotage. Grâce à elle, *" le L4G examine à tout moment l'ensemble des opérations spécifiées de façon déclarative et effectue les opérations se rapportant à l'état dans lequel se trouve le système."* [LAUR.89] Le moteur de la dynamique ne se trouve donc plus au sein d'une procédure, comme en L3G, mais il est éclaté entre tous les objets de l'application, qui précisent eux-mêmes quelle action déclencher lors de leurs changements d'état.

### -a- les objets

L'objet central du pilotage est l'action; c'est une entité générique qui correspond en fait à un F/R, à un menu ou à une procédure.

### -b- les relations entre objets

Il s'agit des liens de déclenchement qui existent entre les objets de l'application (relations, F/R, B/G, champ de B/G, menu, commande, procédure) et les actions possibles.

### -c- paramètres déterminants

*-Complexité, taille et nombre des objets*

- Une action étant soit un F/R, soit un menu, soit une procédure, on se reportera à l'analyse respective de ces objets.
- Un autre critère doit être pris en considération: la fréquence de déclenchement de ces actions, autrement dit le volume de transaction imposé par le SI.

*-Nombre de relations entre objets*

Le nombre de liens de déclenchement entre les actions. Il faut à cet égard tenir compte du fait qu'un appel de procédure depuis une autre procédure peut être fait à l'intérieur d'une structure itérative, et effectuer le comptage des appels en conséquence.

*- Complexité des relations entre objets*

Elle est liée au nombre de paramètres, aux autorisations d'accès, ... que comporte la relation de déclenchement.

Cette modélisation reste fort théorique; elle pose de gros problèmes au niveau de la détection des objets de traitement et de pilotage, parce que la démarche classique d'analyse fonctionnelle ne les aborde pas de façon adéquate. Les traitements perdent avec les L4G leur rôle de pivot central pour ne devenir que des appendices attachés aux objets de la BD et du dialogue. De même, la dynamique de l'application n'est plus centralisée dans un module unique et autonome, mais elle est dispersée au sein de tous les objets de l'application. [LAUR.89] suggèrent une remise en cause de la découpe des phases en fonctions, et proposent de recentrer la conception sur la transaction (création, mise-à-jour, consultation), qui est plus étroitement associé aux objets décrits ici. Cette nouvelle approche est encore au stade expérimental; il n'était donc pas possible de l'appliquer concrètement ici.

## *5. Contraintes retenues:*

### -a- influençant le choix

- complexité, nombre et taille des objets à traiter; nombre et complexité des relations entre ces objets.

Nous nous baserons sur cette typologie pour "mesurer" le SI à développer, et en déduire les spécifications d'une application-test capable de le résumer fidèlement [cfr chapitre 5].

## 3.9. Les autres SI

### *1. compatibilité avec SI existants*

Est-il possible d'échanger des données avec d'autres BD si besoin est ? Est-il possible de coopérer avec certains programmes ou progiciels déjà en exploitation?

En effet, l'entreprise peut exploiter beaucoup d'autres applications ou progiciels, et donc de nombreuses BD. Pour beaucoup de L4G décisionnels (tableurs,...), les principales sources de renseignements sont ces BD générales de l'entreprise, desquelles ils extraient des renseignements filtrés (querries) afin d'en tirer des analyses de synthèse. Un L4G décisionnel sans passerelle vers ces sources d'approvisionnement serait de bien peu de valeur. Il est fréquent que le CTI doive développer cette passerelle qui n'était pas prévue d'origine, ou pas adaptée; c'est un comble quand on sait que le L4G était censé alléger sa charge de travail !

Un L4G orienté développeurs devra aussi tenir compte de tout ce qui existe; il doit s'intégrer dans un environnement exploitant déjà de nombreuses applications qu'il n'est pas question de réécrire immédiatement. La plupart de ces applications sont anciennes (parfois 15 ans !). Le L4G devra coexister, voire coopérer, avec ces anciens systèmes. Il faudra donc composer avec toutes ces BD, parfois hiérarchiques, alors que le L4G est relationnel. Dans certains cas, il faudra parfois faire migrer ces BD anciennes; le L4G le permet-il ?

Le L4G doit également préserver les possibilités d'évolution de l'entreprise, et assurer la compatibilité de ses applications avec un standard reconnu. On n'ose imaginer ce qu'il adviendrait d'une entreprise dont le L4G central serait retiré du marché, sans possibilité d'en faire migrer les applications !

Enfin, certaines entreprises aimeraient voir le même L4G tourner dans tous leurs environnements: mainframe, minis départementaux, réseaux de micros,...

### *2. Contraintes retenues:*

- avec quelles BD et quels programmes devra communiquer le L4G ?

- le L4G supporte-t-il un standard reconnu ?

## 3.10. Pondération des contraintes

Selon l'entreprise ou le département auquel est destiné le L4G, les contraintes évoqués ci-dessus auront plus ou moins d'importance; il conviendra donc de les pondérer [GCM.88]. Cet aspect sera analysé au chapitre 5.

# 4. CRITERES DE SELECTION

Le chapitre 3 a relevé les contraintes imposées par l'environnement. Il convient à présent d'évaluer leur impact sur les caractéristiques du langage à choisir. Nous tenterons d'associer à chaque contrainte de l'environnement la (ou les) fonctionnalité(s) rendant un L4G capable d'y faire face. Ces fonctionnalités, selon qu'elles sont présentes ou pas dans un L4G, témoigneront de son adaptation à l'utilisation qu'on veut en faire. Elles serviront donc de critères de sélection [point 4.2].

Les contraintes imposées par l'environnement ne suffisent pas pour établir un choix définitif. En effet, d'autres critères, indépendants du contexte, ont un grand impact sur la qualité globale du L4G. Ces critères, liés aux qualités fondamentales attendues d'un L4G, seront étudiés sous la rubrique "Critères généraux" [point 4.3].

Nous commencerons par répertorier les différents moyens de mesurer la réponse d'un L4G à ces critères. Nous les avons classés en deux catégories: d'abord les "indices", qui permettent de déterminer rapidement dans quelle mesure un critère est satisfait par un L4G, ensuite les "tests approfondis", qui seront utilisés pour les critères importants ou difficiles à mesurer sans "mettre la main à la pâte".

Comme nous l'avons souligné dès l'introduction, nous veillerons à ne spécifier que des critères généraux; cela permet de garder ses distances vis-à-vis de paramètres trop précis fort liés aux contingences du moment. De même, le choix des moyens d'évaluation des critères est laissé à l'appréciation du décideur, qui devra s'aider d'études récentes, de check-lists permettant de contrôler les qualités d'un bon SGBD, d'un bon report generator,... [MART.85]

Nous établirons enfin une typologie des L4G, groupés par grandes familles traditionnelles. Ce tableau, ainsi que tous les autres panoramas disponibles dans la littérature, seront utiles lors de l'étape de pré-sélection [cfr chapitre 5].

## 4.1 Les moyens de mesure

Les moyens de mesure répertoriés ici sont suffisamment généraux pour s'appliquer à l'ensemble des fonctionnalités d'un L4G. Toutefois, certains critères comme la convivialité, la productivité,... ne peuvent être sérieusement évalués que par des tests ("Hands-on comparison"); nous préciserons en cours de route pour quels critères des tests approfondis s'imposent.

En règle générale, tout réside d'une part dans le coût qu'on est prêt à consentir pour la sélection du L4G, d'autre part dans l'importance relative du critère dans le cadre de décision examiné. On acceptera de consacrer plus de temps à la mesure d'un critère jugé essentiel.

Dans un souci de généralité, nous avons jugé préférable de laisser libre le choix des moyens de mesurer les critères, en fonction de la dépense tolérée pour la procédure d'évaluation, et de la pondération des contraintes imposées par le cadre de décision.

Les critères susceptibles d'une évaluation rapide, basée sur des indices, permettront de "dégrossir" le terrain à peu de frais, d'effectuer une pré-sélection.

Il faudra réserver les critères exigeant des tests au petit nombre de L4G issus de la pré-sélection [cfr chapitre 5].

**-a- les indices:**

Ils permettront de mesurer la réponse apportée par le L4G à un critère, sans devoir effectuer de test approfondi. On les utilisera lors de l'étape de pré-

sélection, afin de déterminer si les fonctions nécessaires sont disponibles, complètes, efficaces,... On pourra utiliser, entre autres:

- des "check-lists" pour vérifier si les fonctionnalités sont complètes et satisfaisantes [MART.85, chapitre 20]

- les applications-test prises comme exemples par le fournisseur (dans les publicités, la documentation ...) Ces applications sont choisies pour vanter le L4G; elles ne s'intéressent donc qu'à ses points forts.

- la documentation des L4G (si on peut y avoir accès)

- l'avis d'autres utilisateurs du L4G [COMP.90, GCM.86]

- les valeurs par défaut des options du L4G indiquant le domaine d'application "par défaut"

- les bancs d'essai et panoramas proposés par les revues spécialisées

- les réponses, démonstrations et tests effectués par le fournisseur sur demande d'un client potentiel

Les critères permettant que cette mesure rapide soit suffisamment fiable serviront de critères de pré-sélection.

### -b- tests approfondis:

Ils seront utilisés pour mesurer les critères qui ne peuvent se satisfaire d'indices rapides. Ils sont basés sur le développement d'applications-test, et la mesure de différents paramètres: temps de développement, de formation, facilité d'utilisation, puissance, performances,...

## 4.2 Critères liés à l'environnement

Nous reprendrons ici les contraintes du chapitre 3 pour en déduire les caractéristiques qu'un L4G doit posséder pour les satisfaire. Certaines caractéristiques sont plus universelles que d'autres: elles satisfont à plusieurs contraintes différentes, dans plusieurs contextes d'utilisation. Dans un souci de clarté et de généralité, elles seront reportées au point 4.3 "Critères généraux".

### 1. L'environnement externe

**-a- l'offre technologique (hard et soft)**

Cette contrainte peut se traduire par les exigences suivantes sur le L4G et la procédure d'évaluation:

- disposer d'études, panoramas, comparaisons récentes, tenant compte des évolutions récentes de l'offre en matière de L4G. Ces études devront également être complètes et tenir compte de l'ensemble de l'offre dans un environnement donné.

- les L4G sélectionnés devront intégrer les idées et approches les plus avancées en ce qui concerne notamment l'utilisation de l'interface graphique, les outils de représentation et de manipulation des objets et structures, ... A cet égard, un L4G doit s'améliorer fréquemment: l'apparition d'une nouvelle version chaque année est chose courante pour les L4G en micro-informatique.

- les L4G doivent également tenir compte des grands standards du marché, surtout en ce qui concerne les fichiers de données; un L4G pour PC-compatible ne permettant pas l'importation/exportation de fichiers au format DBase III, ou un L4G pour mainframe IBM ignorant les fichiers VSAM, est pratiquement condamné d'avance

- les fournisseurs doivent également être surveillés de près: leur solidité, stabilité, service clientèle, le nombre de copies du L4G déjà installées, ... Le taux de décès élevé dans la jungle des L4G ne permet pas de fantaisie quant au choix du fournisseur: sera-t-il toujours là l'an prochain ? Certains petites entreprises sont absorbées dans de grands groupes qui n'assureront peut-être plus le suivi du produit, et qui se soucieront peu de fournir à un trop petit nombre de client les outils de migration du L4G abandonné vers un L4G standard.

### -b- le niveau moyen des connaissances informatiques des utilisateurs visés

Cette contrainte influence la facilité d'emploi attendue du L4G. Ses pré-requis devront être compatibles avec le bagage informatique de l'utilisateur destinataire moyen. Il est très difficile de mesurer le degré de connaissances informatiques des utilisateurs: tel cadre peut avoir "travaillé" quatre ans à l'aide de son tableur Lotus-123 sans pour autant être sorti d'un petit groupe de fonctions utilisées sans trop les comprendre... D'autre part, la facilité d'emploi d'un L4G dépend de nombreux critères qu'il serait vain de vouloir délimiter: convivialité, cohérence de l'approche, simplicité des concepts, intégration et homogénéité des fonctions,... Pour ces raisons, nous pensons qu'un test pratique est indispensable: il faudra confronter l'utilisateur à l'outil, et voir s'il est à même de l'utiliser utilement pour son travail au bout de deux jours (c'est le "two-day test" de Martin).

## 2. L'environnement organisationnel

### -a- la place du L4G parmi les autres outils de développement d'applications

On peut décider d'introduire un L4G dans l'entreprise pour cinq raisons:

-1- pour remplacer totalement le Cobol et les autres outils de développement "lourds". On attend d'un tel L4G d'offrir:

- la puissance: on doit pouvoir faire avec ce L4G tout ce que le Cobol permet de faire; il doit offrir une gamme complète de fonctions de traitement, gestion de données et dialogue.

- la précision de pilotage et la souplesse que permet un L3G; pour ce faire, les automatismes et les choix par défaut, qui procurent une bonne partie des gains de productivité, doivent être débrayables sans devoir y consacrer trop de temps. Un bon composant procédural est également indispensable, permettant de manipuler les mêmes objets que les outils non-procéduraux, mais avec un contrôle plus fin.

- les performances doivent rester satisfaisantes, même avec les gros volumes de transaction que le Cobol supportait.

Le choix de l'outil de développement central de l'entreprise ne peut se faire à la légère; des test et essais pratiques nous semblent indispensables. Il faut que le L4G choisi soit à même de supporter nos ambitions.

-2- pour aider les développeurs dans une étape précise du cycle de développement (construction de prototypes, générateurs de code, langages de

spécification,...), ou dans un domaine de fonctionnalités particulières (gestion BD, générateur d'écrans, ...). Bien que certains auteurs [MART.85, BOUT.84] considèrent ces outils comme des L4G, on peut se permettre d'en douter. Une fois encore, dans la jungle de l'offre en quatrième génération, tout dépend de la définition qu'on en donne. D'après celle donnée au début de ce mémoire, de tels outils sortent du cadre des L4G parce qu'ils s'écartent de la convivialité, de l'intégration et de la non-procéduralité qu'on est en droit d'attendre des L4G. En effet, qu'est-ce qui distingue ces outils des librairies d'utilitaires fournis avec les L3G pour gérer certaines fonctionnalités telles que la BD (SQL pour Cobol ou C), les écrans (TDMS pour Cobol et C, Windows), ou des outils de génie logiciel, qui relèvent d'une approche fort différente de celle des L4G [chapitres 6 et 7] ?

-3- pour offrir aux utilisateurs finals des outils devant les assister dans leurs tâches sans avoir à passer par le circuit de développement traditionnel. On attend de tels outils d'offrir une grande facilité d'emploi (convivialité, cohérence de l'approche, simplicité des concepts, intégration et homogénéité des fonctions, non-procéduralité,...) sans pour autant devoir lui sacrifier trop de potentialités. Un outil rudimentaire est d'office plus simple d'emploi, mais quel intérêt y a-t-il à l'utiliser ?

-4- pour aider dans des domaines d'application spécialisés. On les appelle langages d'application ou progiciels; ils sont plus ou moins paramétrables, mais ne peuvent prétendre qu'à une utilisation bien spécifique. Comme nous l'avons dit dans l'introduction, il semble raisonnable de ne pas étendre la quatrième génération au-delà des systèmes suffisamment polyvalents pour permettre le développement d'une large gamme d'applications.

-5- pour familiariser l'entreprise avec la nouvelle approche que représente la quatrième génération, et tous les changements d'habitude qu'elle impose, avant de généraliser son utilisation. Dans ce cas se pose le problème du coût que l'on peut consentir pour l'acquisition de ce L4G cobaye. Un L4G trop bon marché sera incomplet et peu représentatif, un L4G trop cher

amènera une grosse perte sèche si on décide d'abandonner la voie "quatrième génération".

## -b- le rôle des utilisateurs finals et du CTI dans l'utilisation du L4G

Comme nous l'avons souligné dans l'introduction, les rôles sont attribués par le schéma directeur. Martin distingue trois catégories d'utilisateurs d'un L4G:

- **le développeur**, appartenant au CTI, se borne à écrire du code sans avoir beaucoup de contacts avec l'utilisateur.

- **l'analyste** travaille en collaboration avec l'utilisateur; il l'aide à préciser ses besoins lors de l'analyse fonctionnelle de l'application.

- **l'utilisateur**, dont certains auteurs, trop enthousiastes, attendaient qu'il développe seul ses applications grâce aux L4G. L'expérience actuelle prouve qu'il s'agit là d'une utopie dès que l'application atteint une certaine ampleur. Le cas de Santa-Fé Railroad, dont nous avons déjà parlé, semble être une exception non-reproductible. Dans le cadre d'applications opérationnelles, leur rôle sera sensiblement le même qu'avec les L3G; tout au plus spécifieront-ils plus facilement leurs besoins si l'analyste utilise des outils de prototypage. C'est principalement dans les applications décisionnelles que les L4G leur apporteront les avantages les plus significatifs.

Par rapport à ces trois rôles, on peut distinguer trois situations extrêmes pour le développement d'applications:

- le CTI est développeur, analyste et maître d'oeuvre
l'utilisateur est seulement consulté lors de l'analyse fonctionnelle.

On se situe ici dans des applications opérationnelles, requérant un L4G orienté développeurs. On attend alors du L4G qu'il diminue le coût de développement de deux façons:

- améliorer la productivité
- diminuer la charge de maintenance

Nous renvoyons au point 4.3 pour une analyse plus détaillée de ces critères

- le CTI est un développeur-consultant, qui offre ses services à l'utilisateur; l'utilisateur est maître d'oeuvre et commanditaire; on peut trouver des analystes dans les deux camps.
Ces méthodes plus participatives permettent de diminuer la résistance des utilisateurs, ou de mener à bien une analyse fonctionnelle difficile (besoins flous,...). Pour qu'un L4G soit utile dans le cadre de ces méthodes, il doit faciliter la construction et l'amélioration progressive de prototypes (outils de prototypage ou langage de spécification).

- le CTI assiste de loin en loin, à la demande expresse de l'utilisateur (rôle typique d'un infocentre); l'utilisateur est seul développeur et analyste.
On se situe ici dans des applications décisionnelles, requérant un L4G orienté utilisateurs finals. On attend alors du L4G qu'il décharge le CTI de toutes les demandes ponctuelles des utilisateurs. L'utilisateur devant être le plus autonome possible, la facilité d'utilisation est primordiale. A ce titre, on tiendra compte de critères tels que la convivialité, la simplicité des concepts, la clarté de la documentation et des aides on-line, la guidance et l'accueil, l'homogénéité des fonctions et du dialogue, la non-procéduralité, la continuité entre les modes de développement et d'utilisation,... Nous renvoyons au point 4.3 pour plus de détails sur ces critères.

**-c- la priorité (le délai) accordée au développement du (ou des) SI avec le L4G.**

Un L4G devant développer des applications urgentes devra fournir au développeur une productivité maximale, au détriment de la souplesse et de la polyvalence. Les valeurs par défaut et autres automatismes seront les bienvenus, pourvu qu'ils soient exactement adaptés au cas envisagé. Comme nous le soulignerons dans le chapitre 7, débrayer certains automatismes peut faire perdre énormément de temps.

**-d- but de rentabilité (L4G opérationnel) ou de qualité de la décision (L4G décisionnel) ?**

On retrouve ici la dichotomie traditionnelle dans l'utilisation des L4G, qui sont utilisés soit pour diminuer le coût de développement, soit pour décharger le CTI des demandes ponctuelles des utilisateurs.

On attend d'un L4G opérationnel qu'il améliore la productivité des développeurs, en facilitant notamment la gestion des entrées de données, qu'il diminue la charge de maintenance, qu'il assure de bonnes performances sur de gros volumes de transaction, et qu'il soit assez souple pour permettre de développer la large gamme d'applications dont l'entreprise a besoin.

On attend d'un L4G décisionnel qu'il facilite la gestion des sorties (extraction / manipulation / présentation des données), et qu'il soit facile à utiliser pour un décideur non-informaticien. Une utilisation décisionnelle consistant souvent à rechercher et explorer des solutions (que se passera-t-il si...?), l'outil devra tout faire pour rassurer l'utilisateur et le pousser à la découverte. Dans les cas où aucun extract de la BD ne leur sera fourni par l'infocentre, il faut en outre que le L4G soit directement compatible avec les BD ou SI existants, qui constituent ses réservoirs de données.

### -e- buts flous, donc besoin d'outils d'aide à la spécification (prototypage)

Un prototype sert à concrétiser rapidement les besoins exprimés par l'utilisateur, afin de l'aider à les spécifier de façon précise et définitive. On pourra développer ce prototype de façon incrémentale, en commençant par la partie visible de l'application: le dialogue (écrans et rapports). Une fois le dialogue complètement spécifié, on pourra développer progressivement les fonctionnalités sous-jacentes. Ces méthodes incrémentales, où le prototype est peu à peu raffiné pour devenir l'application finale elle-même, ont fait leurs preuves dans le domaine où l'analyse est probablement la plus périlleuse: le développement de systèmes experts [MOUL.90]. Les L4G avancés, permettant de centrer le développement sur le dialogue, pour y accoler ensuite les traitements, sont tout-à-fait adaptés à ces méthodes de développement.

### -f- la dépense tolérée pour acquérir le L4G

L'évaluation du coût global du L4G doit tenir compte de l'acquisition elle-même, de l'installation et de la "customisation" (adaptation aux besoins précis du client), de l'abonnement aux upgrades et nouvelles versions au fur et à mesure de leur sortie, de la formation et de l'assistance assurée par le revendeur (aide téléphonique via une "hot-line", dépannages...). On obtiendra ainsi le coût global du L4G sur toute sa durée de vie dans l'entreprise.

### -g- l'assistance aux utilisateurs finals

Si aucune structure d'assistance n'est en place (infocentre,...), et qu'on n'envisage pas de le faire pour l'implantation du L4G, on devra veiller à fournir aux utilisateurs un outil dont ils pourront se servir de façon autonome,

sans autre assistance que la documentation, l'aide on-line, et, éventuellement, la hot-line du revendeur. Un L4G facile à installer, apprendre et utiliser sera indispensable. Seul un test réel auprès des utilisateurs concernés nous semble fiable pour vérifier ces qualités.

## 3. L'environnement d'utilisation

### -a- la formation de base de l'utilisateur final, et donc la convivialité attendue de l'outil.

En schématisant beaucoup:

Un "scientifique" préférera un outil puissant, performant, riche en possibilités, qui permette un pilotage précis de toutes ses fonctionnalités. Il n'aura pas peur de la complexité et de la perte de convivialité qu'entraînent de telles exigences.

Un "littéraire" préférera un outil convivial, rassurant, qui le guidera pas à pas dans sa découverte, en masquant les plus possible tous les aspects "techniques".

### -b- la fonction de l'utilisateur final dans l'entreprise

Un décideur travaillera avec un SIAD (système interactif d'aide à la décision), sur des tableaux et extraits de BD préparés par des experts. Vu son peu de disponibilité, il voudra un outil facile à utiliser et totalement adapté à ses besoins. Les utilisations classiques de ces SIAD sont les présentations graphiques et les tests d'hypothèses, en vue de d'analyser des tendances. On parle alors de tableurs. Mais on peut envisager d'autres applications susceptibles d'aider les décideurs: les outils de gestion de projet et

d'ordonnancement, les bases de données documentaires permettant d'archiver et d'accéder à l'historique des décisions prises, par exemple.

Un expert voudra un outil plus complet, lui permettant de préparer les décisions (extraction des données significatives de la BD, préparation des graphiques, des hypothèses, ...).

### -c- la place de l'utilisateur dans le circuit de développement

Elle détermine la philosophie de l'outil (orienté développeurs ou utilisateurs finals). Nous renvoyons au point 2.b du présent chapitre pour plus de détails.

### d- le niveau de décision:

*opérationnelle*--- but de performance; L4G opérationnel, gestion des entrées
*pilotage*---------- niveau intermédiaire, prenant des décisions sur base d'informations issues plus directement des SI opérationnels, sans trop les synthétiser.
*stratégique*------- but de synthèse d'informations; L4G décisionnel, gestion des sorties

Nous renvoyons au point 2.d du présent chapitre pour plus de détails.

### -e- le degré de structuration de la décision :

*structurée* -- but de remplacer l'homme; stabilité, routine, informatique opérationnelle

*informelle* --but d'aider l'exploration; instabilité, non-planifié, informatique décisionnelle

Nous renvoyons au point 2.d du présent chapitre pour plus de détails.

### -f- le domaine d'application

Les L4G les plus courants s'intéressent au développement de systèmes d'information de gestion classiques (opérationnels ou décisionnels), basés sur un SGBD à usage général. Les systèmes scientifiques sont d'office beaucoup plus spécialisés dans un domaine précis, et se rapprochent plus des progiciels que des L4G. Comme nous l'avons dit dans l'introduction, il semble raisonnable de ne pas étendre la quatrième génération au-delà des systèmes permettant le développement d'applications. Un progiciel spécialisé dans la conception assistée (CAD), par exemple, ne sera considéré comme L4G que s'il permet de développer des applications personnalisées; la simple paramétrisation ou "customisation" d'un progiciel aux besoins du client nous semble insuffisante pour lui attribuer le label "L4G".

## 4. L'environnement de développement

### -a- Les méthodes d'analyse utilisées avec le L4G

Les méthodes classique d'analyse semblent devoir être revues, du moins partiellement [voir chapitre 2 et LAUR.89]. Utiliser une méthode d'analyse inadaptée oblige à traduire les spécifications en terme d'objets manipulés par le L4G, ce qui fait fondre le gain de productivité qu'un tel outil peut offrir.

### -b- la place du CTI dans le développement

Elle détermine le type de L4G (développeurs ou utilisateurs finals). Nous renvoyons au point 2.b du présent chapitre pour plus de détails.

### -c- le degré de saturation du CTI

Faut-il attribuer une priorité absolue à la productivité de l'outil, ou préfèrera-t-on ménager l'avenir optant pour un outil favorisant la réutilisabilité et la maintenance des applications, ce qui ne peut s'obtenir qu'au détriment de la productivité immédiate [chapitre 7] ?

## 5. L'environnement opérationnel

### -a- les ressources disponibles

Les L4G sont très gourmands (temps CPU, espace disque,...), et , par leur facilité d'emploi, poussent les utilisateurs à la consommation; les ressources disponibles (CPU, mémoire, disques, réseau, opérateurs, ...) supporteront-elles ce surcroît de demande ? Le coût de l'upgrade éventuel du matériel et de l'infrastructure est à mettre en balance avec le gain en productivité du développement. L'inflation des accès aux données pose aussi des problèmes de sécurité du réseau, de maintien de l'intégrité des données, ... Si des postes d'administrateur de données ou de surveillant du réseau n'existent pas encore, l'introduction d'un L4G offre une bonne occasion de les créer.

En outre, les L4G ne sont pas égaux face aux problèmes de performance [voir point 4.3.6 et JALI.89].

**-b- la richesse de l'environnement d'exploitation**

L'environnement exploité dans le centre où doit être implanté le L4G détermine la gamme de langages envisageables. Certains environnements sont très fournis en L4G (MS-DOS, Macintosh, MVS), d'autres ont moins de chance.

# 6. Le(s) SI à développer avec le L4G

Nous nous baserons sur la caractérisation des SI en fonction des nouveaux concepts apportés par les L4G [voir chapitre 3].
Rappelons qu'un ordre de grandeur des applications à développer sera souvent suffisant pour départager les L4G.
Il faudra évaluer les L4G passés en revue selon deux axes:

- **quantitatif**: les L4G étudiés supportent-ils ces volumes, cette complexité, ces relations ? Ont-ils une carrure suffisante ?

- **qualitatif**: les L4G étudiés offrent-ils des outils supportant tous les types de primitives dont on peut avoir besoin pour définir, gérer et utiliser ces objets ? Ces outils pourront en outre être analysés sur base des critères généraux de convivialité, non-procéduralité, productivité,...

Nous allons reprendre ici les principales primitives de définition, gestion et utilisation d'objets et relations entre objets qu'un L4G devrait posséder.

### -a- la Base de Données

- Primitives de création et de gestion de la structure de la BD, généralement via un dictionnaire des données.
- Primitives d'accès aux données: création, consultation, destruction, modification de records vérifiant un filtre (avec index éventuels).
- Primitives de contrôle des données: validation de la création, suppression, insertion ou modification d'un champ ou d'un record, sur base des contraintes d'intégrité et des autorisations d'accès qu'il doit vérifier.

### -b- le dialogue

- Primitives de création et de gestion des objets du dialogue: F/R (comportant des champs de B/G et des boutons), menus et commandes.
- Primitives d'utilisation des objets du dialogue: lier les objets du dialogue équivaut à spécifier la dynamique de l'application; bien que les liens entre les objets doivent être définis à l'intérieur des objets eux-mêmes, on préférera centraliser ces aspects au sein de la fonction de pilotage.
- On retrouve en outre les primitives de validation de la saisie, qui peuvent être spécifiées indistinctement sur la BD ou sur les F/R.

### -c- les traitements

- Primitives de création et de gestion des traitements: éditeur de procédures, d'expression calculées, d'opérations d'agrégation.
- Primitives d'utilisation des traitements: appliquer une procédure ou un expression sur certaines parties de la BD, selon un filtre,...

### -d- le pilotage

- Primitives de création de liens de déclenchement entre les actions (F/R, menus, procédures), permettant de définir la dynamique de l'application.

Ces liens de déclenchement seront activés en fonction des changements d'état des objets (aussi appelés "**événements**"):

- pour la BD: avant, après l'insertion, la modification, la suppression d'un record ou d'un champ.

- pour le dialogue : à l'activation ou désactivation d'un F/R, d'un B/G, d'un record de B/G, d'un champ, d'un menu, d'un bouton; au choix d'un item de menu, à l'introduction d'une commande, avant ou après un query.

La plupart des événements sont automatiquement pris en charge par le L4G, mais peuvent aussi être détectées "manuellement" (par des primitives du langage procédural) pour un contrôle plus fin.

- Primitives de détection des événements, de déclenchement d'une action et de contrôle de l'enchaînement, utilisées à l'intérieur de procédures.

# 7. Les autres SI

### -a- échanges avec SI existants

Est-il possible d'échanger des données avec d'autres BD, SI ou progiciels si besoin est ? Les échanges de données doivent se faire dans les deux sens: du L4G vers les autres SI, et vice-versa. Ils peuvent se faire par extraction de fichier à fichier, ou par communication on-line. Le L4G retenu devra donc accepter les formats de données des S.I. avec lesquels il entrera en relation (VSAM, SQL et pour la micro-informatique, DBase III, Lotus 123, les grands

traitements de texte,...). En outre, un L4G sur micro-ordinateur devra impérativement comporter des fonctions d'échange via le réseau du mainframe.

## 4.3 Critères généraux

Dans ce paragraphe, nous nous intéresserons aux qualités générales qu'un L4G doit posséder, au moins en partie, pour mériter cette appellation. Outre cette "moyenne" qu'un bon L4G doit respecter, les contraintes de l'environnement mettront l'accent sur telle ou telle qualité, jugée primordiale dans un cadre de décision précis. Par exemple, si la convivialité est une des vertus fondamentales attendue d'un L4G, indépendamment de tout contexte, un environnement centré sur l'utilisateur final rendra ce critère plus important que d'habitude. Pour schématiser, ces critères généraux seront gratifiés, lors de la pondération, d'une valeur par défaut (représentant la moyenne), augmentée de l'importance attribuée par les contraintes de l'environnement à ce critère.

Les qualités générales reprises ici sont réparties en deux catégories:

-a- les qualités fondamentales des L4G, celles qui les différencient des autres environnements de développement. Ces qualités servent d'ailleurs souvent dans les définitions des L4G [BOUT.84, MART.85].

- convivialité
- non-procéduralité
- intégration
- productivité
- facilité de maintenance

-b- les critères suivants font référence aux points faibles traditionnels des L4G, qu'il faut surveiller de près:

- performances
- souplesse et flexibilité
- sécurité et fiabilité
- lisibilité, clarté de la structure d'ensemble

Ces qualités pourront servir à évaluer chacun des grands groupes de fonctionnalités du L4G: BD, dialogue, traitements et pilotage.

## 1. convivialité

*"La convivialité d'un outil logiciel, c'est littéralement la possibilité de vivre avec lui, avec tout ce que cela peut impliquer en termes de confiance, dialogue, fidélité, disponibilité, durabilité, aisance et même plaisir, etc."* [BOUT.84]

De nombreuses études ont été faites sur l'aspect "human factoring" des logiciels. Les remarques et règles d'or présentées ici sont extraites de [BODA.89.b] Elles s'adressent, dans le cadre qui nous occupe, au L4G lui-même, mais il faudra également en tenir compte lors du développement des applications: on peut développer une application rébarbative à l'aide du L4G le plus convivial qui soit.

On distingue généralement deux "fossés" entre le monde de la tâche de l'utilisateur et la représentation qu'en fait l'outil logiciel qu'il utilise:

- la **distance sémantique** est l'écart qui existe entre les objectifs de l'utilisateur et la signification des expressions du langage; peut-on exprimer facilement ce qu'on veut réaliser dans le langage ?

- la **distance articulatoire** est l'écart qui existe entre la signification des expressions du langage et leur forme; peut-on déduire facilement de la forme d'une expression sa signification ?

Une bonne interface-utilisateur devra s'attacher à réduire ces deux distances.

Plus concrètement, on évalue les qualités d'une interface à l'aide de **"règles d'or"** [SHNE.88]:

- lutter pour la cohérence: terminologies et séquences de commandes identiques pour des situations identiques.

- fournir de l'information en retour à l'utilisateur, le guider pour qu'il sache toujours où il est et où il va. L'aide on-line et la bonne structuration de la documentation sont importants à cet égard.

- structurer le dialogue en tâches fermées, pour donner le sentiment d'avancer réellement vers la solution.

- offrir une gestion d'erreurs: détection rapide, aide à la correction

- donner à l'utilisateur l'impression qu'il pilote le système; éviter les séquences d'opération rigides, sur lesquelles l'utilisateur n'a pas de prise.

- réduire la charge de mémoire de l'utilisateur: le transfert d'informations dans le système humain est d'autant plus rapide que le recodage est réduit. Des concepts simples et provenant directement de l'univers de la tâche de l'utilisateur seront maîtrisés sans effort. Dans le même ordre d'idées, on veillera à la concision.

- offrir plusieurs représentations du même concept, selon la tâche à accomplir (représentations sous forme de menu, de graphique, de texte...)

L'interface devra également tenir compte de la **diversité des utilisateurs**:

- les **novices** sont anxieux; il faut donc les rassurer et les inviter à découvrir les possibilités de l'outil utilisé. On leur fournira en particulier la possibilité de défaire les opérations erronées qu'ils ont accomplies, pour leur donner le sentiment que rien n'est irrémédiable dans ce qu'ils font.

- les **utilisateurs intermittents** ont du mal à maintenir leur connaissance de l'outil; on pourra, par exemple, leur rappeler les commandes au sein de menus cohérents.

- les **utilisateurs expérimentés** dominent l'outil et souhaitent aller plus vite dans son utilisation; les profondes arborescences de menus, la "verbosité" de l'interface sont autant de pertes de temps qu'ils voudraient éviter. On leur permettra, par exemple, de "débrancher" les modes d'utilisation trop "verbeux" ou trop automatiques, d'utiliser des raccourcis-clavier pour la sélection d'items de menus,...

De façon plus générale, l'interface doit pouvoir s'adapter à l'évolution de la maîtrise de l'utilisateur sur l'outil; le dialogue sera structuré en couches de complexité croissante.

Des tests pratiques peuvent être menés afin de déterminer les qualités de l'interface. On pourra mesurer:

- le temps d'apprentissage des commandes nécessaires à l'exécution d'une tâche
- la rapidité d'exécution de ces commandes
- le taux d'erreurs de manipulation et leur temps de correction
- la satisfaction subjective: le confort, l'enrichissement, la facilité de découverte,...
- la période de rémanence: la simplicité de l'interface et sa cohérence par rapport à l'univers de la tâche de l'utilisateur.

On comprendra aisément que d'autres critères comme la non-procéduralité, qui permet de manipuler directement les concepts et objets de la tâche, ainsi que la cohérence et l'intégration de ces concepts, jouent un rôle déterminant dans la convivialité d'un L4G.

## 2. non-procéduralité

La programmation procédurale impose de spécifier pas à pas la séquence d'opérations élémentaires à effectuer pour arriver au but recherché.

La programmation non-procédurale prend de la hauteur par rapport au détail des opérations; il suffit de spécifier le but à atteindre, sans se préoccuper du cheminement opératoire.

Dans le cas des L4G, l'utilisateur n'aura qu'à spécifier l'objectif qu'il veut atteindre, l'outil se chargera lui-même des détails de mise en oeuvre. La non-procéduralité des L4G est de type déclaratif: il suffit d'exposer le but à atteindre sous la forme de spécification d'un ensemble de faits. Il existe d'autres types de non-procéduralité: basée sur des règles logiques (systèmes experts), sur des modèles non-algorithmiques (tableurs),...

Les principales conséquences de la non-procéduralité sur les L4G sont:

- la convivialité, la guidance profitant aux utilisateurs finals, qui ne s'embarrasseront pas de détails techniques de mise en oeuvre.

- un pilotage moins direct de la machine, une perte de flexibilité et une restriction du domaine d'application, car beaucoup de décisions sont prises par le L4G, beaucoup moins souple et adaptable que le programmeur humain. La non-procéduralité est une étape supplémentaire dans la course vers les langages de haut niveau.

- moins de code: seuls les buts à atteindre sont spécifiés; ils sont toujours beaucoup plus concis que les opérations de réalisation qu'ils engendrent.

- moins de "bugs"; les erreurs arrivent lorsqu'on doit spécifier une séquence rigoureuse d'opérations précises. L'esprit humain s'accommode mal des raisonnements séquentiels et rigoureux imposés par la machine,

où chaque détail a des conséquences multiples et difficilement prévisibles. L'exigence de rigueur sera moins contraignante dans la spécification des buts, puisqu'ils sont plus proches du raisonnement humain.

- la maintenance est facilité par la légèreté et la clarté du code; une description de buts est plus lisible qu'une séquence d'opérations de bas niveau.

- l'outil soulage l'homme des opérations de mise en oeuvre; la productivité du développeur sera donc améliorée, pour autant que l'outil non-procédural soit tout-à-fait adapté à l'application à développer (ou adaptable à peu de frais). Il est rare qu'une application sérieuse rentre totalement dans ce qui était prévu par le L4G. C'est pourquoi le recours à un langage procédural est souvent inévitable. Selon Martin, la meilleure productivité est atteinte lorsque le L4G dispose d'un mode non-procédural (pour la productivité) et d'un mode procédural (pour la flexibilité).

Le dosage de procéduralité et de non-procéduralité nécessaire pour développer une application à l'aide d'un L4G donné est un bon indice de la puissance de ce L4G et de son adaptation au type d'application demandé. Une étude estime que le code procédural ne doit pas dépasser 25% du total. Tout la difficulté réside dans l'estimation de ce dosage; voir [VERN.88].

Les implications de la non-procéduralité seront analysées au chapitre 6.

# 3. intégration

## -a- intégration des objets

Les L4G peuvent être découpés en leurs quatre composantes principales: la base de données, le dialogue, les traitements et le pilotage. La force des L4G

est de permettre au développeur d'utiliser les objets dont il a besoin exactement où il en a besoin. Un champ d'une relation de la BD, un bouton de pilotage, une procédure,... peuvent être placés directement sur le F/R qui les utilisera. Le cohérence de cet ensemble est assurée par le dictionnaire des données, qui centralise la description des objets de l'application et de leurs relations.

Les outils de manipulation de ces objets doivent aussi être cohérents: les primitives procédurales et les outils non-procéduraux s'efforceront d'offrir les mêmes possibilités et la même approche dans leur manipulation des objets [voir point 4: productivité].

Certains L4G, basés sur un noyau de fonctionnalités pré-existantes (SGBD, L3G, gestionnaire d'interface-utilisateur), et "gonflés" pour parvenir au statut de L4G, n'offriront pas cette cohérence d'ensemble; certaines limitations trahiront toujours leur "philosophie" d'origine.

### -b- intégration des fonctionnalités

Les L4G les plus récents offrent, au sein d'un structure cohérente, des fonctionnalités que l'on ne trouvait autrefois que dans des L4G différents, avec toutes les difficultés que cela impliquait en terme de différences de syntaxe, d'échange des données,... A titre d'exemple, un des L4G les plus avancés dans ce domaine (4ème Dimension pour Macintosh) offre cette vaste palette de fonctionnalités: extracteur et manipulateur de données, tableur, générateur de graphiques, traitement de texte, SGBD, langage de programmation, générateur de documentation des applications,...., s'ajoutant, bien sûr, à sa fonction centrale: le générateur d'application. Ce degré ultime d'intégration permet de lier dynamiquement les objets d'une application avec tous ces outils décisionnels; toute modification d'une donnée dans la BD se répercute automatiquement dans le tableau et le texte qui l'utilisent. Certains, comme Guru (sur PC), vont même jusqu'à offrir un moteur d'inférence et une base de connaissances.

# 4. productivité

Les gains de productivité que permettent les L4G s'expliquent par plusieurs de leurs caractéristiques:

- la puissance de leurs outils non-procéduraux. Un L4G bien adapté au domaine d'application permettra d'aller directement à l'essentiel de la tâche de développement; prenant beaucoup de décisions à la place du développeur (valeurs par défaut, prise en charge automatique d'événements,...), il le débarrassera des détails fastidieux.

- Comme on l'a souligné plus haut, malgré la puissance des outils non-procéduraux, il restera toujours des cas "tordus", face auxquels ces outils seront démunis. Deux solutions sont alors possibles:

  -On parvient à paramétrer l'outil non-procédural en "débrayant" certains automatismes ou valeurs par défaut.

  - Si ce débrayage est impossible ou insuffisant, il faudra pouvoir pallier aux déficiences de l'outil non-procédural en programmant soi-même, sans pour autant devoir renoncer entièrement à la puissance qu'il offrait. D'où l'intérêt de disposer de primitives reproduisant la partie intéressante des fonctionnalités de l'outil non-procédural abandonné, tout en permettant un réglage plus précis des divers paramètres. En outre, ces primitives devront être totalement cohérentes avec l'outil non-procédural dans leur façon de manipuler les objets de l'application. Cela évitera de devoir perdre trop de temps à comprendre leurs différences d'approche. Le langage procédural utilisant ces primitives devra être aussi soigné que la partie non-procédurale du L4G: on devra y retrouver toutes les fonctionnalités et structures de contrôle qui font la force des L3G actuels. Hélas, trop de L4G n'offrent qu'un langage procédural artisanal, souvent bien en-deçà des possibilités des L3G actuels.

- d'autres critères jouent un grand rôle dans la productivité des L4G: la facilité de maintenance, l'intégration de fonctionnalités autrefois dispersées dans plusieurs L4G, la convivialité, ...

## 5. facilité de maintenance

Le bilan dans ce domaine est mitigé: certains aspects des L4G facilitent la maintenance, alors que d'autres semblent la rendre plus compliquée.

Les caractéristiques **positives** sont:

- le dictionnaire des données qui centralise dans une structure claire et cohérente toutes les descriptions des objets composant l'application.

- la diminution du volume final de code

- le code non-procédural est plus lisible: il suffit de comparer la description d'un formulaire de saisie en Cobol (sous forme d'une séquence de primitives) et en 4ème Dimension (où il est créé à même l'écran).

- certains L4G répercutent automatiquement sur tous les objets reliés les modifications faites sur un d'entre eux dans le dictionnaire de données. Ainsi, la destruction d'un champ d'une relation fait disparaître ce champ des F/R et qui l'utilisent. Il en va de même pour le changement du type d'un champ. Les choses vont beaucoup moins bien dès qu'on aborde les traitements: la destruction d'un champ dans une relation BD n'entraîne que l'effacement de son nom dans les procédures qui l'utilisent. Le L4G serait en effet bien incapable d'évaluer plus avant les répercussions que cet effacement amènent sur la procédure dans son ensemble. Tant qu'on se borne à des primitives simples (affichage, saisie, affectation d'une valeur,...), on peut se satisfaire de la simple disparition du nom du champ;

mais imaginons par exemple que le champ effacé fasse partie de la condition d'une structure itérative...

Cette idée de propagation automatique des changements atteint son sommet dans les tableurs, où toute modification du contenu d'une case se répercute automatiquement sur toutes les cases liées par formule. Cela est rendu possible par le caractère purement mathématique des objets (cellules) et des liens entre eux (formules). En ce qui concerne un L4G, les objets et les liens entre eux n'ont pas cette simplicité.

Les caractéristiques dont l'effet sur la charge de maintenance est **mitigé**:

- l'éclatement des traitements et de la dynamique, dispersés entre les différents objets qui les utilisent [voir point 9: clarté-lisibilité].

## 6. performances

On reproche souvent aux L4G leur gourmandise, aussi bien en temps CPU qu'en nombre d'entrée/sortie disque, en volume disque occupé,... Quelles raisons peut-on trouver à cela ? Quelles solutions peut-on y apporter ?

- Leur SGBD est relationnel, ce qui entraîne une dégradation des performances (temps CPU, nombre d'entrées/sorties disque, espace disque) par rapport à une BD hiérarchique [SUCH.88]. Un SGBD relationnel est non-procédural; on ne spécifie pas les chemins d'accès aux données, le SGBD les choisit lui-même, ce qui peut amener à des requêtes mal optimisées. Un SGBD hiérarchique est plus procédural: on doit lui décrire de façon précise les chemins d'accès aux données, ce qui permet au développeur de choisir lui-même le chemin optimal pour une requête donnée.

- Les L4G, de par leur convivialité et leur vocation "utilisateur final", poussent à la consommation et entraînent généralement une inflation de la demande qui

peut, si on n'en tient pas compte, laisser croire que le L4G dégrade les performances de la machine.

- Plus un langage de programmation s'éloigne du code machine pour se rapprocher des spécifications, plus il est amené à prendre des décisions lui-même en ce qui concerne les détails de la mise en oeuvre. Une application décrite sous forme de ses spécifications (schéma E/A, statique et dynamique des traitements et dialogues) est beaucoup plus concise que si elle était décrite sous forme de langage procédural de bas niveau. Dans le premier cas, l'outil de développement aura à prendre beaucoup de décisions pour pallier aux nombreux sous-entendus et non-dits des spécifications. Dans le second cas, le développeur devra traduire pas à pas les spécifications en instructions exécutables; généralement, il produira un code plus performant, plus optimisé, que s'il était produit par un programme: la programmation reste un art qui demande une qualité de jugement que la machine n'a pas.

- Une des approches en L4G consiste à fournir à l'utilisateur un outil lui permettant de développer lui-même ses applications décisionnelles (consultations BD, rapports,...). Cela permet de décharger le CTI des petites demandes ponctuelles des utilisateurs. Mais un utilisateur final ne formulera pas sa requête d'une façon aussi optimale qu'un développeur professionnel. Il pourra, par exemple, consulter un fichier énorme sur base d'une clé non-indexée, effectuer des tris sans index,... Bref, un L4G rend les utilisateurs capables de saturer complètement le système. Pour éviter cela, il serait intéressant de prévoir, avant son exécution, le temps que prendra une requête, d'en avertir l'utilisateur et d'empêcher les requêtes trop gourmandes.

- L'optimisation des performances doit être possible, au moins pour les parties critiques de l'application. On y arrivera en débrayant certains automatismes lourds, en se passant d'options inutiles, voire en programmant soi-même certains modules critiques.

- Un L4G permettant de compiler les applications avant de les transférer dans l'environnement de production permettra d'améliorer sensiblement les

performances par rapport à l'exécution interprétée, généralement utilisée dans l'environnement de développement afin de faciliter le debugging.

On gardera toujours à l'esprit que la dégradation des performances, et donc la surconsommation de ressources qu'elle entraîne, est à mettre en balance avec les gains de productivité du développement que les L4G apportent. La productivité est améliorée si on utilise directement les facilités offertes par le L4G, sans les retoucher. Cela implique généralement une dégradation des performances, qui, à leur tour, peuvent être améliorées au prix d'une dégradation du coût de développement.

Enfin, il faut reconnaître que la gourmandise des L4G arrange bien les constructeurs, qui ont toujours "poussé à la consommation" en matière de L4G; ils peuvent ainsi écouler des upgrades ou des machines plus puissantes.

Une étude récente [JALI.89] montre que les L4G ont bien évolué; les meilleurs L4G (sur PC) atteignent maintenant le niveau de performances du Cobol. Cependant, un L4G est souvent très performant pour un petit nombre de fonctions, et déplorable pour les autres (parfois 200 à 700 fois plus lents!). On notera toutefois qu'il est difficile de comparer un outil qui ne fait rien sans qu'on le lui demande (Cobol), avec des outils qu'on pilote plus qu'on ne les programme (les L4G); souvent, ils prendront des décisions, invisibles au niveau du développeur, qui fausseront les comparaisons. Par exemple, certains L4G créent systématiquement des index cachés, devenant ainsi plus performant que le Cobol qui, si on ne lui demande pas d'ouvrir un index, ne le fera pas.

Même si le Cobol reste globalement plus performant que n'importe quel L4G, ces derniers ont accompli des progrès rendant leur réputation de lenteur quelque peut obsolète.

## 7. souplesse et flexibilité

La puissance des outils non-procéduraux entraîne, nous l'avons dit, une restriction du domaine d'application du L4G. Toute application imposant, pour son développement, de sortir de ce domaine provoque immanquablement une perte de productivité, tantôt parce qu'il faut adapter ces outils non-procéduraux en débrayant certains de leurs automatismes, tantôt parce que, ces outils étant totalement inutilisables, il faut programmer soi-même en langage procédural.

Deux questions se posent dès lors:

- le domaine d'utilisation des outils non-procéduraux est-il suffisamment large ? Sont-ils suffisamment paramétrables pour permettre au développeur de les adapter à ses besoins (valeurs par défaut modifiables, automatismes débrayables,...) ?

- au cas où les outils non-procéduraux sont inutilisables, le L4G dispose-t-il d'un bon langage procédural qui, tout en offrant les mêmes possibilités de manipulation d'objet que les outils non-procéduraux, permette un contrôle plus précis des opérations. Pour ce faire, les fonctionnalités atomiques significatives de chaque outil non-procédural devraient se retrouver sous forme de primitive du langage procédural.

- Malgré la présence d'un langage procédural puissant et complet, la plupart des L4G sont limités au domaine des applications de gestion de données. Des applications scientifiques, liées à l'automation, à la surveillance de processus en temps réel, aux simulations de modèles complexes,... sortent du cadre des L4G. On leur préférera pour de telles situations des L3G ayant fait leurs preuves, permettant de décrire des algorithmes plus complexes et plus précis, offrant de meilleures performances, et disposant d'impressionnantes librairies de fonctions : C, ADA, Modula-2, Fortran...

Pour contourner leurs limitations, les L4G permettent généralement d'incorporer des modules écrits et compilés dans d'autres langages. Cela ne suffit bien sûr pas pour faire des L4G des outils adaptés au calcul scientifique, mais ils peuvent ainsi gérer certaines fonctionnalités scientifiques ou techniques nécessitées par l'application, sans obliger le développeur à trop "bricoler". Par exemple, en micro-informatique, des fonctionnalités telles que le pilotage de caisses enregistreuses, de lecteurs de code-barre,... sont réalisées de cette façon.

La flexibilité peut se comprendre au sens plus large de portabilité des données, voire des applications elles-mêmes, d'un environnement opérationnel à l'autre. Ce problème se pose de deux façons:

- plusieurs environnements coexistent au sein de l'entreprise; on voudrait les rendre compatibles: mainframe, mini, réseaux de stations de travail et de micros. Certains L4G sont disponibles dans plusieurs environnements différents.

- l'entreprise veut garder la possibilité de pouvoir changer d'environnement opérationnel (matériel et système d'exploitation), ou de L4G. La migration des applications développées dans l'ancien environnement ou avec le vieux L4G est-elle possible ? Beaucoup de fournisseurs de hardware et software préfèrent limiter ces possibilités de façon à lier définitivement leur clientèle à leurs produits. Le taux de décès élevé dans le monde des L4G impose la plus grande méfiance à l'égard des outils coupés de tout standard.

## 8. sécurité et fiabilité

- Les "bugs" sont en théorie moins nombreux car le code est réduit; des problèmes peuvent toutefois se poser car il faut comprendre tous les effets des outils offerts. Plus ils sont de haut niveau, plus ils cachent des interactions au

développeur, qui peut passer des heures à découvrir certains "effets de bord" non-prévus, ou mal expliqués dans la documentation.

- à cela s'ajoute souvent un manque de clarté de la structure d'ensemble (éclatement des traitements,...), rendant indispensable un bon outil de debugging ou de suivi pas à pas des opérations.

- Les L4G entraînent souvent une augmentation du nombre d'utilisateurs connectés. La confidentialité est-elle assurée à tous les niveaux: accès aux données, opérations autorisées, ... ?

- Dispose-t-on de bonnes possibilités de prévention et de récupération des crashes ? Ces crashes sont-ils fréquents ? Imposent-ils un arrêt total du L4G et de toutes les applications sous son contrôle ?

## 9. lisibilité, clarté de la structure d'ensemble

- Le dictionnaire de données centralise les descriptions des objets de l'application et les structure autour des objets de la BD (les relations). Dans un L3G, ces descriptions étaient noyées dans le listing des différents modules fonctionnels de l'application. Dans un L4G, la modularité est centrée, non plus sur les fonctionnalités de l'application, mais sur les relations de la BD. Cela a pour effet de rendre plus facile la localisation des descriptions d'objets de la BD, bien sûr, mais aussi du dialogue (les F/R) et des traitements (procédures), qui sont définis sur les relations BD. Si cette structuration est bénéfique pour la localisation des objets, elle amène en revanche un éclatement des fonctionnalités et de la dynamique de l'application.

Cette subordination des traitements aux objets de la BD et du dialogue rend pénible la maintenance ou le debugging d'une fonctionnalité, dispersée à travers tous les objets qu'elle met en oeuvre. Quels moyens nous permettraient de pallier à cet éclatement ? La génération automatique d'une documentation centralisée sur les traitements, l'adoption de conventions imposant de lier tel type de traitement à tel type d'objet,... [voir chapitre 6].

## 4.4 Panorama des L4G d'après les critères proposés

Ce classement des types de L4G par grandes fonctions permettra de se rendre compte, dès le début de la procédure de choix, si un seul outil suffira. Une trop grande diversité dans les contraintes relevées indiquera au contraire que les besoins sont multiples, et qu'un seul L4G ne suffira pas.

On verra également vers quel type de L4G s'orienter, avant même la pré-sélection (cfr chapitre 5 pour le détail de la procédure de sélection).

Nous ne proposons ici qu'un état général (et incomplet) de l'offre, classée par grandes catégories; pour des analyses plus détaillées des potentialités de ces outils, on se référera aux essais, panoramas, comparaisons de la littérature et des revues spécialisées [BOUT.84, MART.86 a et b, ROUX.88, SVM, BYTE...]. Les progiciels, vu l'étendue de l'offre, ne seront pas repris ici; de même, on exclut tous les outils spécialisés dans une partie du développement, qui nous semblent plus relever d'une approche "génie logiciel" (langages de spécification, langages de programmation de très haut niveau, outils de prototypage, gestion BD, générateur d'écrans, .....).

Nous reprenons ici les classifications des L4G proposées par Martin [MART.85] et Boutel [BOUT.84], quelque peu hétéroclites, mais qui reflètent bien l'éparpillement de l'offre en matière de L4G.

On reprendra la distinction majeure entre les L4G opérationnels et les L4G décisionnels:

Les L4G destinés aux applications **décisionnelles** privilégieront l'accès aux données déjà stockées, leur présentation, bref, les outputs. Etant souvent

destinés aux utilisateurs finals, ils devront être faciles d'emploi. On peut les classer comme suit:

- les **interrogateurs**, permettant d'extraire des données de la BD.

- les générateurs de **rapports** de synthèse qui présentent les données par agrégats.

- les langages de **manipulation** de BD par les utilisateurs: saisie, validation, mise-à-jour, production et stockage de nouveaux résultats, mais aussi interrogations complexes, permettant de naviguer dans la BD.

- les systèmes d'aide à la décision (**SAD**) : tableurs, processeurs de modèles, outils de simulation, synthèse des données brutes avec présentation graphique,...

Un L4G destiné aux applications **opérationnelles** privilégiera le développement d'applications nouvelles (structuration, saisie, validation des données), bref, les inputs. On attend d'eux d'améliorer la productivité du développement et de faciliter la maintenance. On peut les classer comme suit:

- les **progiciels** paramétrables, qui offrent une solution "clé sur porte" dans de nombreux domaines (gestion comptable, financière, commerciale, gestion de projets et ordonnancement, mais aussi automation, CAD/CAM, support de R&D, ...). Ils sont plus ou moins paramétrables, mais ne peuvent néanmoins prétendre qu'à une utilisation bien spécifique. L'adaptation du progiciel aux besoins de l'utilisateur (et à leur évolution), extrêmement complexe, est souvent assurée par le fournisseur.

- les **générateurs** d'applications permettent de développer complètement une application; cette catégorie recouvre en fait les L4G les plus avancés, privilégiant le non-procédural, la prise en charge automatique des événements,...

- les outils centrés sur un aspect particulier du développement: la **base de données** (les SGBDR comme DBase III) ou le **dialogue** (gestionnaires d'interface comme Hypercard). Nous ne reprenons toutefois pas dans cette catégorie les nombreuses "toolbox" permettant d'étendre les possibilités d'un L3G vers la gestion de fichiers ou du dialogue; ces outils relèvent plus du génie logiciel que de L4G autonomes, conviviaux et non-procéduraux.
En dehors de leur point fort, ces outils montrent vite leurs faiblesses: ils doivent alors être couplés à d'autres outils (souvent des L3G) permettant de pallier à leurs lacunes. C'est ainsi que DBase III se révèlera vite insuffisant dans le domaine du dialogue; cela explique la profusion d'outils voulant améliorer DBase sur ce point. De même, Hypercard montrera vite ses faiblesses en gestion de données. Dans le même ordre d'idées, on notera que beaucoup de L4G "complets" sont originaires d'un domaine bien précis (souvent la gestion de BD). Certains ont été mal "élargis" à d'autres domaines, et trahissent vite leur spécialité d'origine.

Nous classerons ces outils dans les 8 grandes catégories définies ci-dessus. Nous nous sommes intéressés aux outils pour mainframes **IBM**, pour mainframes **non-IBM,** pour PC IBM ou compatibles, et pour **Macintosh.**

| | L4G décisionnels | | | | L4G opérationnels | | | | Mainframe | | Micro | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Outils** | query | report | manip | SAD | génér applic | progic | BD | dialog | IBM | Non IBM | PC | Mac |
| ADF | X | | | | X | | | | X | | | |
| ADN | X | X | | | X | | | | | | | X |
| ADRS II | X | X | X | X | | | | | X | | | |
| ADS | | | | | X | | | | | X | | |
| Alpha Four | | | | | X | | | | | | X | |
| Application *f*actory | X | X | | | X | | | | | X | | |
| Application Builder | | | | | X | | | | X | | | |
| AS | X | X | X | X | | | | | X | | | |
| CA-Universe | | | | | X | | | | X | | | |
| Clarion Professional Developer | | | | | X | | | | | | X | |
| Condor 3 | X | X | X | | | | | | | | X | |
| CSP | X | | | | X | | | | X | | | |
| D the data language | | | | | X | | | | | X | X | |
| Data Boss | | | | | X | | | | | | X | |
| DATabase | X | | X | | | | | | | | | X |
| Dataease 4.0 | X | X | | | X | | | | | | X | |
| Datascope | X | X | | | X | | | | X | | | |
| Datatrieve | X | X | X | | | | | | | X | | |
| DBase III+ | X | X | | | X | | | | | | X | |
| DBase IV | X | X | | | X | | | | | | X | |
| DBase Mac | X | X | | | X | | | | | | | X |
| Directory II | X | X | X | | | | | | | | X | |
| DMS | | | | | X | | | | X | | | |
| Double Helix | X | X | X | | | | | | | | | X |
| Easytrieve | X | X | | | | | | | X | | X | |

| | L4G décisionnels | | | | L4G opérationnels | | | | Mainframe | | Micro | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Outils** | query | report | manip | SAD | génér applic | progic | BD | dialog | IBM | Non IBM | PC | Mac |
| Excel | | | | X | | | | | | | X | X |
| Express | | | | X | | | | | X | | | |
| EZT Plus | X | X | X | | | | | | X | | | |
| File 2.0 | X | | X | | | | | | | | | X |
| File Force | X | X | X | | | | | | | | | X |
| File Vision IV | X | | X | | | | | | | | | X |
| Filemaker II | X | X | X | | | | | | | | | X |
| Focus | X | X | X | X | X | | | | X | | X | |
| FoxBase / Foxpro | X | X | | | X | | | | | | X | X |
| Framework | X | X | X | X | | | | | | | X | |
| Full Impact | | | | X | | | | | | | | X |
| Gener /OL | X | X | | | X | | | | X | | | |
| GIS | X | X | | X | | | | | X | | | |
| Hypercard | X | | | | | | | X | | | | X |
| IC /1 | X | X | X | | | | | | X | | | |
| Ideal | X | X | | | X | | | | X | | | |
| IDMS /R | | | | | X | | | | X | | X | |
| Info | | | | | X | | | | X | X | | |
| Informix | | | | | X | | | | | X | X | |
| Inquire | X | X | | | | | | | X | | | |
| Intellect | X | X | | | | | | | X | | | |
| Knowledge Man /2 | X | X | X | X | | | | | | | X | |
| Linc | X | X | | | X | | | | | X | | |
| Lotus 123 | | | | X | | | | | | | X | |
| Mapper 10 | X | X | | | X | | | | | X | | |
| Mark V | X | X | | | X | | | | X | | | |
| MC Mac | | | | | X | | | | | | | X |

| | L4G décisionnels | | | | L4G opérationnels | | | | Mainframe | | Micro | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Outils** | query | report | manip | SAD | génér applic | progic | BD | dialog | IBM | Non IBM | PC | Mac |
| Millenium Application / SDT | X | X | X | X | | X | | | X | | | |
| Mimer | X | | | X | X | | | | X | X | | |
| Multiplan | | | | X | | | | | | | X | |
| Natural | X | X | | | X | | | | X | | | |
| Nomad 2 | X | X | X | X | X | | | | X | | | |
| Omnis 5 | X | X | X | | X | | | | | | | X |
| Open Acces | X | X | X | X | | | | | | | X | |
| Oracle | X | X | X | X | X | | | | | X | X | X |
| Panorama | X | X | X | | | | | | | | | X |
| Paradox 3.0 | X | X | X | | X | | | | | | X | |
| PC Magic | | | | | X | | | | | | X | |
| Personal Data Query | X | X | | | | | | | | X | | |
| PlanPerfect | | | | X | | | | | | | X | |
| QBE | X | | X | | | | | | X | | | |
| QMF | X | X | | | | | | | X | | | |
| Quatrième Dimension | X | X | X | X | X | | | | | | | X |
| Quattro pro | | | | X | | | | | | | X | |
| Quickbuild | X | X | | | X | | | | X | | | |
| R:Base | X | X | X | | | | | | | | X | |
| Ramis II | X | X | X | X | X | | | | X | | | |
| Rapid File | X | X | X | | | | | | | | X | |
| Reflex + | X | X | X | | | | | | | | X | |
| SAS | X | X | X | X | | | | | X | X | | |

| | L4G décisionnels | | | | L4G opérationnels | | | | Mainframe | | Micro | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Outils** | query | report | manip | SAD | génér applic | progic | BD | dialog | IBM | Non IBM | PC | Mac |
| SPSS | | | | X | | | | | X | X | | |
| SQL | X | X | X | | | | X | | X | X | X | |
| Stairs | X | | X | | | | | | X | | | |
| Super DB | X | X | X | X | X | | | | | | X | |
| Supercalc 5 | | | | X | | | | | | | X | |
| Supra / Mantis | X | X | X | | X | | | | X | | X | |
| Sybase | | | | | X | | | | X | | X | |
| System W | | X | | X | | | | | X | X | | |
| TIF | X | X | X | | X | | | | X | | | |
| TIS | X | X | X | | | | | | X | | | |
| Twin | | | | X | | | | | | | X | |
| UFO | | | | | X | | | | X | | | |
| Umbrella System | | | | | X | | | | X | | | |
| Uniface | X | X | | | X | | | | X | | X | |
| Universal Base Six | X | X | X | | | | | | | | X | |
| VP-Planner plus | | | | X | | | | | | | X | |
| Wingz | | | | X | | | | | | | | X |
| Works | X | X | X | X | | | | | | | X | |

# 5. SELECTION

Les chapitres 3 et 4 ont mis en place des outils facilitant la prise de décision [relevé des contraintes de l'environnement, des critères de sélection, panorama des L4G]. Il faut maintenant articuler ces outils dans le cadre d'une méthode de sélection rigoureuse, réaliste et utilisable.

## 5.1 Préciser le cadre de décision

Le système de contraintes relevé au chapitre 3 est général et théorique; il ne se réfère à aucun environnement d'utilisation réel. Il faut donc instancier ce système général en attribuant à ses contraintes une pondération dépendant des caractéristiques de l'environnement où se prend la décision. On va ainsi préciser les paramètres de comparaison (type d'utilisateur, applications à tester,...) et déterminer quels critères sont importants, en leur attribuant une pondération.

Un exemple de méthode de pondération peut être trouvé dans [GCM.86]: la pondération 1 à 1. Des méthodes d'analyse multicritères plus complexes peuvent être utilisées. Elles consistent souvent en une comparaison par paires des critères, qu'on agrège ensuite de façon à établir un ordre de préférence global, ou classement des critères en fonction de leur importance. L'utilisation de telles méthodes oblige parfois à comparer l'incomparable: préfère-t-on un L4G convivial à un L4G acceptant le standard DBase ? Néanmoins, elles aident les décideurs à structurer leurs préférences, et permettent de classer les contraintes par catégories de priorité.

On obtiendra finalement un tableau des critères attendus du L4G, ainsi que leur importance, qui constitueront le cahier des charges de l'outil.

## 5.2 Pré-sélection

### *-a- cibler le type de L4G*

On va d'abord se référer au panorama classant les différents types de L4G par grandes fonctions. En le comparant aux critères du point 5.1, on pourra cibler précisément le type de L4G dont on a besoin.

### *-b- premier tri grâce aux critères "immédiats"*

Certains critères peuvent être mesurés facilement, grâce à des "indices". Cette évaluation rapide et peu coûteuse permettra d'effectuer un premier tri parmi les L4G disponibles.
Les critères exigeant des tests approfondis pour être évalués sérieusement seront réservés aux quelques L4G issus de la pré-sélection. Leur évaluation pour tous les L4G disponibles serait en effet trop coûteuse et inutile dans la mesure où un coup d'oeil rapide suffit souvent pour se rendre compte que beaucoup de ces L4G sont totalement inadaptés.

Comme le fait remarquer F.G. Roux [ROUX.88], l'offre de L4G est devenue tellement foisonnante que beaucoup d'entre eux se tiennent de très près. Pour les départager, il ne suffit plus de cocher les cases demandant si telle ou telle fonction est couverte par le L4G; il faut aussi être nuancé et apprécier la façon dont elle est couverte (très bien, bien, mal).

## 5.3 Sélection

Il s'agit ici de départager les deux ou trois L4G issus des pré-qualifications. L'évaluation des critères sur lesquels on va se baser ici exige de mener des tests. On spécifiera ces tests en tenant compte de:

- la modélisation générale du SI à développer (chapitre 3), afin d'étudier le L4G sous un aspect quantitatif: supporte-t-il les volumes, la complexité, les objets et fonctionnalités qu'on lui demande ?

- les critères précis qu'on veut mesurer (chapitre 4), c'est-à-dire l'aspect qualitatif: les outils offerts par le L4G sont-ils conviviaux, productifs, non-procéduraux, intégrés, performants, souples,... ?

On développera ensuite ces tests, et on appréciera la façon dont le L4G satisfait aux critères demandés, en fonction des remarques émises au chapitre 4.

# PARTIE II : EXPERIMENTATION

Afin de tester la méthode proposée dans ce mémoire, et de vérifier si elle est utilisable et réaliste, l'idéal aurait consisté en une évaluation des besoins de quelques environnements organisationnels concrets et représentatifs, pour ensuite appliquer la sélection sur toute la gamme de L4G disponibles pour cet environnement. Cela n'a pas été possible dans le cadre de ce mémoire. En effet, trouver et évaluer des environnements organisationnels réels, évaluer la totalité de l'offre de L4G dans un environnement donné, demandait un temps considérable pour être mené sérieusement.

Il nous a toutefois semblé indispensable d'étayer les allégations de la partie théorique, basées uniquement sur des lectures, par une confrontation réelle avec quelques L4G représentatifs.

La démarche suivie dans ce chapitre consiste à évaluer trois outils représentatifs des trois grandes catégories de L4G:

- les L4G orientés utilisateurs finals et applications décisionnelles: FileMaker 4, de Nashoba Systems

- les L4G traditionnels orientés développeurs et applications opérationnelles: FoxBase+ de Fox Software

- les L4G avancés orientés développeurs et applications opérationnelles: 4ème Dimension d'ACI.

Nous avons choisi l'environnement Macintosh parce qu'au caractère déjà pionnier de la micro-informatique en matière de L4G, il ajoute une facilité d'emploi très en avance sur les autres environnements disponibles.

Nous les avons évalués dans les deux grands types d'utilisation possible:

- une utilisation décisionnelle par un utilisateur final
- une utilisation opérationnelle, par un développeur.

# 6.Expérimentation

Le but n'étant pas d'évaluer la méthode, mais plutôt de lui donner une assise expérimentale, nous ne l'avons pas appliquée complètement. En particulier, il nous a semblé absurde de tenter de modéliser un environnement organisationnel fictif pour les seuls besoins de cette expérience; nous avons donc abandonné toutes les contraintes organisationnelles (environnements 1.2 à 1.6). Nous avons préféré travailler à un niveau plus général, sur les deux grandes classes d'utilisation possible des L4G et les principales contraintes qu'elles leur imposent.

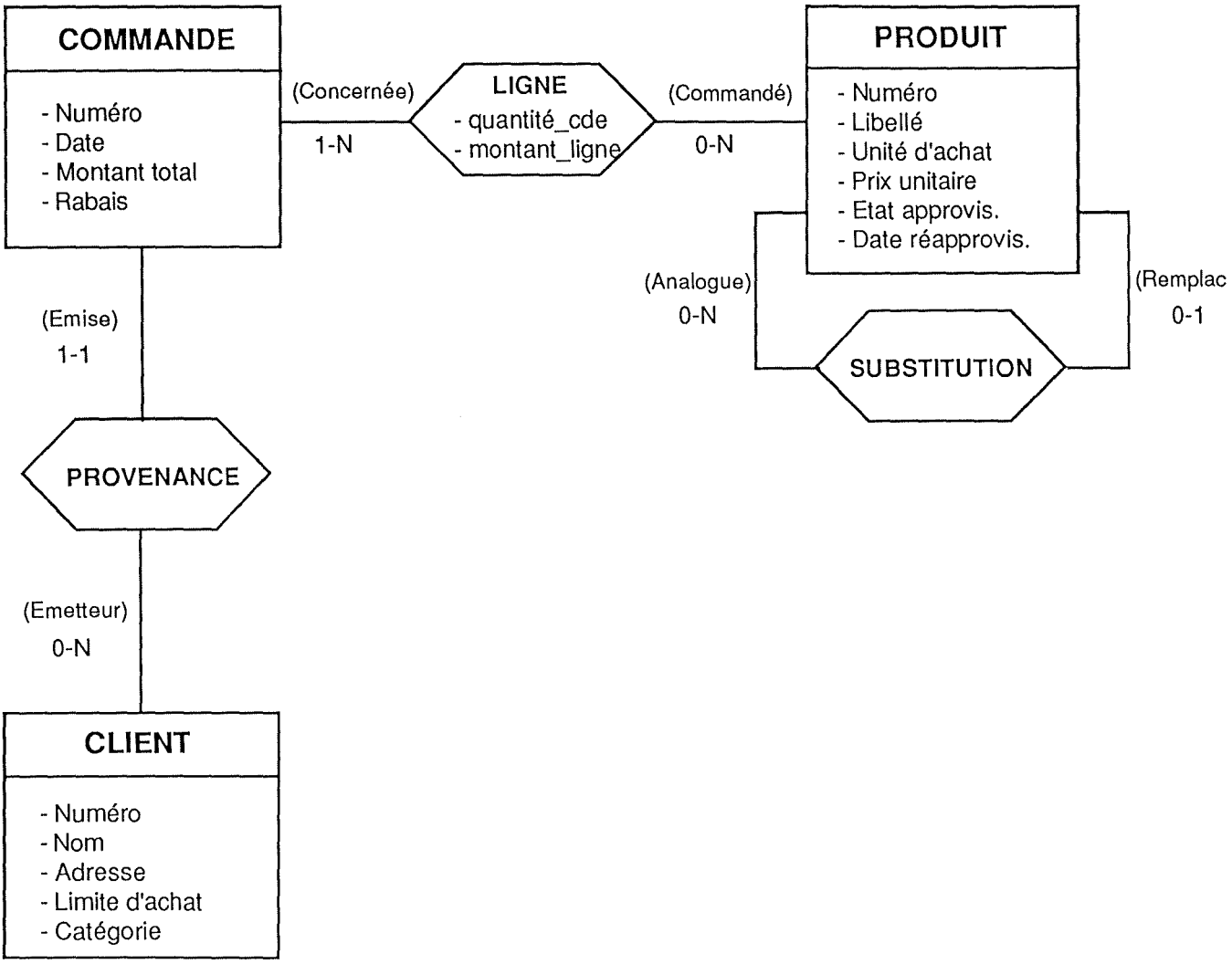
Pour représenter chacune de ces deux classes, nous avons choisi une ou plusieurs applications; nous en exposons brièvement les spécifications et nous tenterons de les évaluer sur base du canevas proposé au chapitre 3.

## 6.1 spécifications des deux types d'utilisation

### A Utilisation opérationnelle

Développement d'une application totalement opérationnelle, centrée sur la gestion des entrées, par un développeur professionnel. Nous nous sommes restreints à une seule phase: le célèbre cas d'école: "saisie des commandes_client". Nous reprenons sommairement ses spécifications; pour plus de détails, voir [LAUR.89].

*Schéma Entité-Association de la phase Commande-Client*




L'objectif de cette phase est de mémoriser une commande, mais aussi, vu les contraintes de connectivité spécifiées, sa provenance (client) et ses lignes.

Pour une description plus précise des fonctions de la phase, des contraintes d'intégrité, de la statique et de la dynamique des traitements, on se reportera à [LAUR.89].

Nous allons à présent utiliser le canevas du chapitre 3 afin de "prendre les mensurations" de cette application, pour chacune des grandes catégories d'objets.

# 1. la Base de Données

- Nombre d'objets

La BD se compose de 4 relations: Client, Produit, Commande, Ligne_commande.

Le nombre d'occurrences (records) de chaque relation de la BD:

10000 clients
15 produits
1000 commandes
1200 lignes_commande

- Complexité des objets

Nombre de champs et de clés indexées pour chaque relation:

Clients: 11 champs, dont un indexé (numéro)
Produits: 7 champs, dont un indexé (numéro)
Commandes: 5 champs, dont un indexé (numéro)
Lignes_commande: 4 champs, dont un indexé (commande_concernée)

- Taille des objets

La taille de chaque relation équivaut à la somme des tailles de ses champs:

Client: 170 bytes
Produit: 72 bytes
Commande: 40 bytes
Ligne_commande: 32 bytes

- Nombre de relations entre objets

Il existe 12 contraintes d'intégrité entre les relations (dont 3 seront traduites sous forme de champs calculés; il s'agit des dépendances fonctionnelles).

- Complexité des relations entre objets

La complexité des contraintes d'intégrité; aucune contrainte n'impose d'accès à plus de deux relations différentes.

## 2. le dialogue

- Nombre d'objets

L'application décrite ici compte 3 F/R; la saisie des lignes_commande se faisant au sein du F/R de la commande.

- Complexité et taille des objets

Le formulaire de saisie d'une commande comporte 2 B/G; un centré sur la commande (11 champs), et un autre sur la ligne de commande (4 champs répétitifs).
Le formulaire de saisie d'un client, ainsi que celui d'un produit, ne comportent qu'un B/G reprenant l'ensemble des champs des relations BD correspondantes.

## 3. les traitements et le pilotage

Un problème pratique se pose en ce qui concerne ces objets: leur spécification sous forme des fonctions d'une phase étant incompatible avec l'implémentation physique qui doit en être faite dans le L4G. *"l'architecture physique des traitements de la phase ne peut être déduite de manière systématique (correspondance 1-1) de l'architecture logique. Cela signifie, entre autres, que les caractéristiques des procédures de la conception logique ne sont pas*

*semblables à celles des actions de la conception physique."* Une autre démarche est proposée dans le mémoire d'où est tirée cette citation [LAUR.89], mais n'a pas été appliquée à l'analyse de la phase "commande-client" qui nous occupe; nous n'avons donc pu en bénéficier ici.

Les traitements et actions étant "éclatés" au sein des objets du dialogue, *"il n'existe pas de règles systématiques de transformation des contraintes d'intégrité ou des règles de traitement en actions ou expressions"* [LAUR.89], d'autant qu'il existe de grosses différences d'un L4G à l'autre: l'un offrira une gestion automatique de certaines contraintes d'intégrité, l'autre obligera au contraire à les programmer "à la main". Il faut donc se baser sur la description logique des traitements, et les répartir entre les objets de l'application d'après les possibilités du L4G. Les méthodes de développement classiques (modèles de la statique et de la dynamique des traitements, modularisation et hiérarchisation) n'offrent pas de critères de répartition physique des traitements et des actions; nous avons toutefois choisi de rapprocher le contraintes d'intégrité et les traitements le plus près possible des objets qu'ils modifient: une contrainte ou un traitement sur un champ BD doit être attaché à ce champ dans le dictionnaire de données. Un problème se pose dès qu'un traitement modifie plusieurs objets différents: au quel rattacher le traitement ? Faut-il répartir le traitement entre tous ces objets ? Mais alors, comment garantir que l'enchaînement sera respecté ?
En ce qui concerne les actions (fonction de pilotage), elles se placent le plus souvent sur l'objet qui amènera l'événement déclencheur (bouton, validation,...). Le même problème se pose: où placer une action susceptible d'être déclenchée par un changement d'état de plusieurs objets ?

La **méthode suivie ici** pour développer cette application reprend les conseils de [LAUR.89].

*1. Définir les relations de la BD, leurs champs et le maximum de contraintes d'intégrité et de champs calculés que le L4G permet d'introduire dès ce niveau (certaines, trop complexes, ne pourront être spécifiées qu'à l'étape des traitements:5).*

*2. Chaque objectif de la phase constitue en fait une transaction sur les données de la BD (création, mise-à-jour ou consultation); pour chaque transaction, nous créons:*

*3. Le F/R qui la représente*

*4. La définition des B/G de ce F/R et de leur hiérarchie*

*5. Les traitements qu'elle impose*

*6. Lier les transactions entre elles: répartir la dynamique*

## B Utilisation décisionnelle

Utilisation simple des outils non-procéduraux fournis par le langage (querries, rapports, graphiques..), sans intégration de ces opérations dans une application, sans programmation procédurale. Bref, une utilisation décisionnelle par un utilisateur final, basée sur la gestion des sorties.
Trois opérations seront effectuées sur base des données de l'application opérationnelle définie ci-dessus:

1- une extraction de données sur base d'un critère complexe, mettant en oeuvre des champs de plusieurs relations: "donner le nom de tous les clients habitant Wavre qui ont commandé pour plus de 1000 francs de tarte".

2- un rapport: "calculer le chiffre d'affaire, sur tous les produits, par région d'habitation du client émetteur (sur base du premier chiffre du code postal)."

3. une présentation graphique de ce rapport.

## 6.2 Présentation des L4G testés

Ici encore, le temps manquait pour permettre d'évaluer la totalité de l'offre en matière de L4G. Nous nous sommes limités à l'environnement qui, tant par sa facilité d'emploi que par l'originalité de son approche en matière de L4G, nous a semblé le plus avancé: le Macintosh. Nous avons restreint la gamme de L4G analysés à un seul représentant pour chacune des trois grandes catégories analysées dans l'introduction.

### A Les L4G orientés utilisateurs finals et applications décisionnelles

La caractéristique essentielle de ces L4G est la convivialité. L'utilisateur est accueilli, guidé, assisté, aussi bien en mode d'exploitation qu'en mode de développement. Le mode non-procédural est privilégié de bout en bout; certains L4G de cette catégorie, et c'est le cas de FileMaker 4, ne disposant même pas d'un langage de développement procédural. L'utilisation décisionnelle se situe au bout de la chaîne de traitement informatique; elle s'occupe donc principalement de la gestion des sorties.

Le représentant de cette catégorie sera FileMaker 4, de Nashoba Systems.
C'est un L4G monofichier, ne fournissant aucun moyen de développer une application personnalisée. Il n'offre que des possibilités rudimentaires d'automatisation des tâches répétitives (les scripts).

## B Les L4G traditionnels orientés développeurs et applications opérationnelles

Les L4G appartenant à cette catégorie privilégient la puissance de l'outil de développement. On les appelle "traditionnels" parce qu'ils gèrent toute l'application par des procédures, comme le font les L3G.

Le représentant de cette catégorie sera FoxBase+ de Fox Software. C'est un outil de développement basé sur un SGBD relationnel, entièrement inspiré de DBase III+, avec lequel il est totalement compatible, aussi bien au niveau des formats de données (.dbf) que des formats d'applications (.prg). Au-dessus de cette couche SGBD, FoxBase ajoute une panoplie complète d'outils non-procéduraux permettant de créer les formulaires d'affichage ou de saisie, les procédures, les relations de la BD,... Il offre en outre un pseudo-dictionnaire des données (le mode "view") qui centralise les descriptions d'objets mais ne permet pas de se passer des procédures pour développer une application complète. La procédure reste le point de passage obligé du développement, caractéristique propre aux L4G traditionnels.

## C Les L4G avancés orientés développeurs et applications opérationnelles

Ils sont identiques à ceux de la catégorie précédente, si ce n'est qu'ils permettent de répartir la dynamique de l'application entre les différents objets qui la composent, ôtant ainsi aux procédures leur caractère central. Ils disposent en outre d'une fonction de pilotage, chargée de gérer et de contrôler les interactions spécifiées entre les objets (dynamique de l'application).

Le représentant de cette catégorie sera 4ème Dimension d'ACI. Il fait partie des outils de développement les plus avancés qui existent sur micro-ordinateur. On retrouve chez lui toutes les caractéristiques habituelles d'un L4G opérationnel: un SGBD relationnel, le mode d'utilisation non-procédural est privilégié, fournissant un maximum d'automatismes, un langage procédural puissant, complet et bien structuré... Son statut de L4G avancé est dû à son mode "structure", véritable dictionnaire de données centralisant autour des relations de la base de données tous les objets de l'application: formulaires de saisie et de consultation (formats), procédures (formules), liens entre deux champs ,...

## 6.3 Critères de sélection des L4G

Nous reprendrons ici les principaux critères de sélection que les deux types d'utilisation choisis imposent.

### A Utilisation opérationnelle

Nous attendons du L4G qu'il offre:

- une bonne productivité
- la facilité de maintenance
- de bonnes performances, même sur des volumes de donnée importants
- une souplesse suffisante pour ses outils non-procéduraux (automatismes débrayables, valeurs par défaut modifiables,...)
- un langage procédural bien structuré, puissant, offrant les mêmes fonctionnalités et la même approche des objets que les outils non-procéduraux, mais avec un contrôle plus précis. Tout cela pour éviter

d'avoir à "bricoler" si tel ou tel cas un peu "tordu" (hors des possibilités des outils non-procéduraux) se présente.

### B Utilisation décisionnelle

Nous attendons du L4G qu'il offre:

- la facilité d'emploi: convivialité, cohérence de l'approche, simplicité des concepts, intégration et homogénéité des fonctions

- l'ouverture aux standards existants, et principalement au standard DBase III.

## 6.4 Résultats des tests

## FileMaker 4 (version 4.0, 1988)

### A Utilisation opérationnelle

Deux caractéristiques de FileMaker 4 le rendent totalement inadapté au développement d'applications sérieuses: il est monofichier (travail sur un seul fichier à la fois) et il n'offre pas de possibilités sérieuses de pilotage ni de traitement (pas de langage procédural, ni de gestion de la dynamique). L'application opérationnelle n'a donc pu être développée sur cet outil. Tant qu'on se limite à des applications simples de gestion de fichiers uniques,

FileMaker 4 est bien adapté; il a accepté sans problème de gérer notre fichier de 10000 clients, avec des performances correctes.

### B Utilisation décisionnelle

FileMaker 4 accepte sans problème les fichiers au format DBase III, après leur transformation au format SYLK par un utilitaire standard du Macintosh: Apple File Exchange. Il accepte également de recevoir des données via un réseau.

FileMaker n'acceptant aucune opération multifichier, l'utilisation décisionnelle s'est soldée, elle aussi, par un échec. Même si l'on se limite à travailler sur un seul fichier, l'outil montre vite ses faiblesses: les requêtes multicritères, les descriptions de rupture et de variables d'agrégation dans les rapports, ne sont pas d'une utilisation très agréable; il permet d'incorporer des dessins aux champs d'un ficher, mais pas de graphes statistiques (histogrammes,...).

Il est assez décevant qu'un outil ne permettant qu'une utilisation décisionnelle soit aussi pénible à manipuler. Dès qu'on aborde des opérations non-triviales, et pourtant élémentaires pour une utilisation décisionnelle (requêtes multicritères, rapports,...), cet outil perd toute la facilité d'emploi qu'il semblait permettre à première vue. Les manipulations deviennent moins évidentes à comprendre, obligeant à procéder par essai et erreur; c'est d'autant plus frustrant que la facilité d'emploi est vantée comme la qualité essentielle de cet outil. Quand on compare FileMaker à 4ème Dimension, on se rend compte que ce dernier gagne sur tous les tableaux: non seulement en utilisation opérationnelle, mais également en utilisation décisionnelle. Un outil disposant de fonctionnalités limitées n'est pas nécessairement plus simple à utiliser qu'un outil plus riche, pourvu que ce dernier soit structuré en couches de complexité croissante, autorisant ainsi plusieurs niveaux d'utilisation [voir 4ème Dimension, ci-après].

Nous en déduisons donc que FileMaker, ainsi que la plupart des autres gestionnaires de fichiers uniques, sont inadaptés pour des utilisations décisionnelles sérieuses. On les utilisera plutôt dans des applications simples telles que l'archivage ou le mailing (un fichier client lié à un traitement de

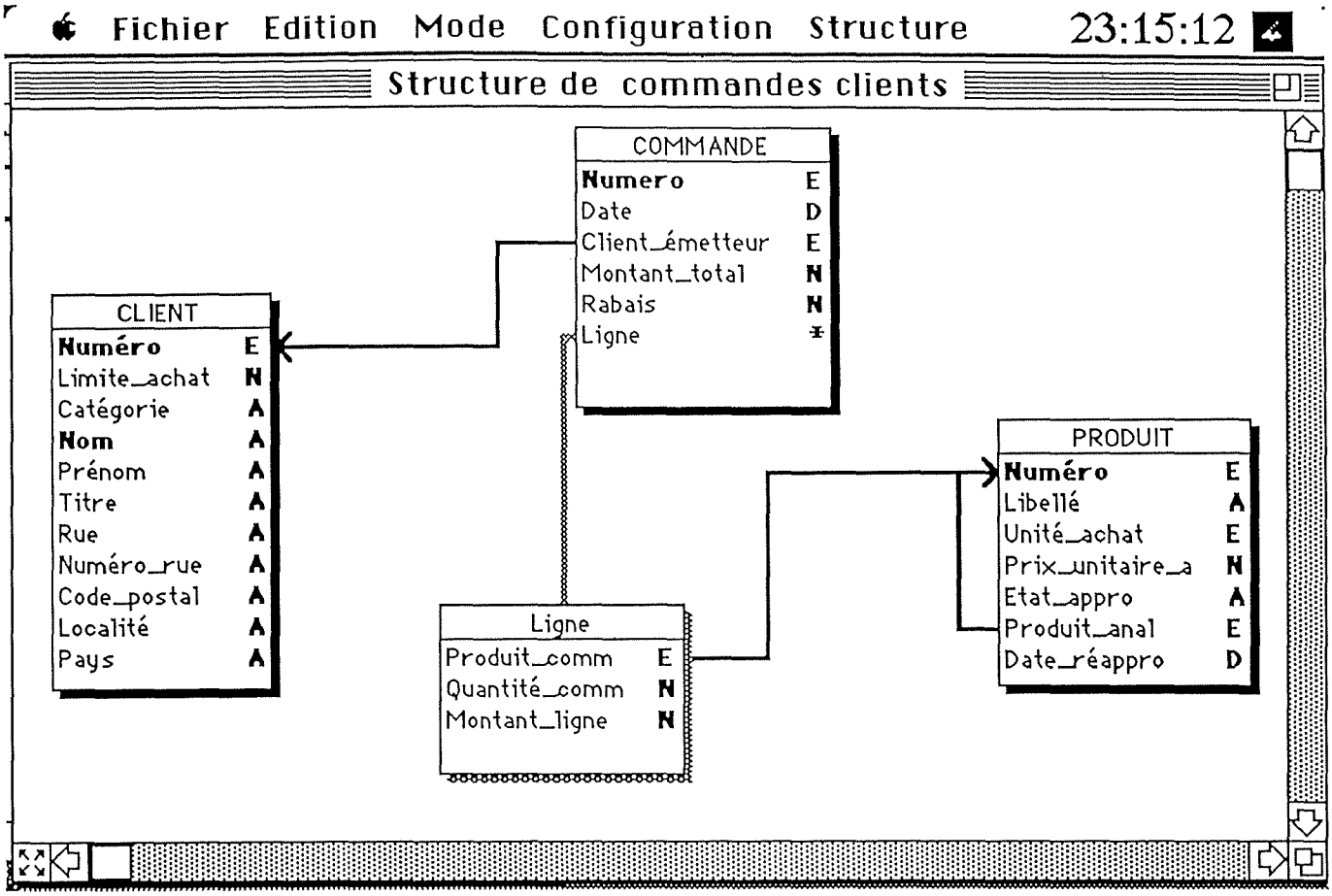
texte). Les possibilités graphiques évoluées de certains d'entre eux les rapprochent plus d'outils de dialogue évolués tels que Hypercard, mettant plus l'accent sur les objets et applications graphiques interactives (le concept de "Multimédia") que sur la gestion sérieuse de fichiers de données, au sens classique du terme.

## 4ème Dimension (version 3.2, 1987)

Les critiques émises ici doivent être relativisées: ce logiciel, dans sa version actuelle (4.0.8) , semble avoir beaucoup évolué par rapport à la version testée ici.

### A Utilisation opérationnelle

Le mode structure de 4ème Dimension (4D) correspond à la notion de dictionnaire des données; il centralise autour des relations de la BD les objets du dialogue (formats) et du traitement (formules-fichier). C'est le tableau de bord de l'outil.

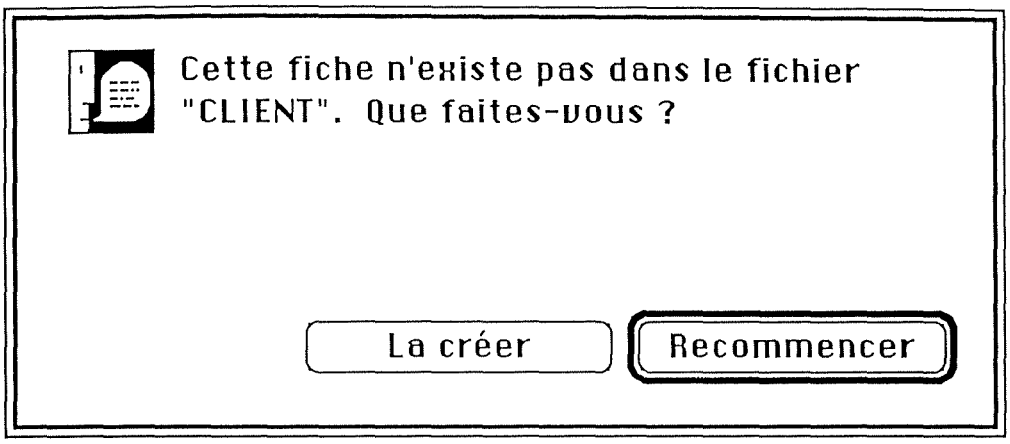



*dictionnaire de données en 4ème Dimension: le mode structure*

## - *productivité*

4D facilite le développement d'applications grâce aux nombreux automatismes qu'il permet:

- un lien établi entre deux champs de même type permet de vérifier automatiquement les contraintes d'intégrité de référence; un message d'erreur explicite avertira l'utilisateur qui essaie, par exemple, de lier une commande à un client inexistant. Un lien peut même être établi entre deux champs d'une même relation (cfr produit_analogue et numéro_produit).




*message automatique avertissant qu'un contrainte d'intégrité de référence n'a pas été respectée: on a voulu attribuer une commande à un client inconnu*

Un lien permet en outre de créer une donnée agrégée (appelée "vue" en SQL), qui prélève des champs dans différentes relations et les lie à un champ-pivot.

- une contrainte d'intégrité d'identification est vérifiée automatiquement; un message d'erreur explicite avertira l'utilisateur qui essaie, par exemple, d'attribuer à un client un numéro d'identification qui existe déjà.

- les champs obligatoires devront être remplis, sinon le record ne sera pas validé.

- la création d'un formulaire de saisie ("formats" en 4D) peut être immédiatement dérivée de la description de la relation sur laquelle il se base; on peut en outre y intégrer facilement des champs d'autres relations liées à la principale; ces champs secondaires ne seront pas modifiables dans le formulaire (on retrouve ici la notion de Bloc/Groupe de [LAUR.89]).

- la représentation des items décomposables et répétitifs est assurée par la notion de sous-fiche en 4D. Dans l'exemple qui nous concerne, les lignes de commande ont été représentées par des sous-fiches de la relation commande. Cela apporte beaucoup de facilités au niveau de la création de

formulaires de saisie (les lignes, sous forme de tableau, sont représentées sur le même formulaire que leur commande-mère), des fonctions d'agrégation, de la consultation,...

## - *facilité de maintenance*

Les formules de 4D se situent à un niveau intermédiaire entre les procédures classiques des L3G et les procédures-appendice des objets telles que définies dans [LAUR.89]. Ces formules sont liées aux seuls objets de haut niveau: relations BD et F/R du dialogue; pour les lier à des objets de plus bas niveau comme les champs, il faut recourir, à l'intérieur de la formule, à des primitives du langage procédural: "si modifie (*nom_de_champ*)". Les formules ainsi obtenues sont hybrides:

- non-procédurales dans leur lien avec l'objet de haut niveau (relation BD ou F/R)

- procédurales dans leur lien avec les objets de bas niveau (champs)

De cette façon, on s'écarte de la philosophie "orienté-objets" qui fait la force de L4G, mais, d'un autre côté, on limite l'éclatement des traitements qui serait maximal si une formule pouvait être liée à chaque champ. Les formules restent ainsi suffisamment cohérentes et globales pour permettre d'utiliser les méthodes classiques de développement et de maintenance. Tant que nous ne disposerons pas d'une méthode de conception d'applications réellement adaptée aux L4G, et donc de critères, de conventions, permettant de répartir les fonctions de l'analyse logique sur les objets qui les utilisent, il est indispensable que les traitements restent d'un niveau suffisamment global. Sans quoi les décisions de répartition prises sans critère ni méthode lors de la conception ne pourront que difficilement être retrouvées lors de la maintenance. On ne facilite jamais la tâche d'un "mainteneur" lorsqu'on l'oblige à se poser la question "qui fait quoi sur qui ?" pour tous les objets qu'il doit modifier... Une fois cette méthode et ces critères définis, il faudra que les L4G en tiennent compte. Par exemple, à la découpe "verticale", orientée-objets (relations BD,

F/R du dialogue, formules des traitements) qu'ils préconisent, ils devraient superposer une découpe "horizontale", basée sur les fonctionnalités de l'application. Cette double découpe permettrait de "tracer" une fonctionnalité à travers tous les objets qu'elle utilise pour arriver à son objectif, réintroduisant ainsi une vue cohérente des traitements et de la dynamique de l'application.

## *- performances*

Elles restent correctes, même sur des volumes de donnée importants (10000 fiches clients). Des essais plus poussés (100000 fiches) ont été menés sur 4D qui a très bien supporté le choc [SVMA.90.9]. En ce qui concerne l'espace disque occupé, le fichier ASCII contenant les 100000 fiches (10 Méga-octets) s'est transformé, après importation sous 4D, en base de données de 70 Mo !

## *- langage procédural*

- le langage procédural a permis de résoudre des problèmes que les outils non-procéduraux ne prenaient pas en charge: entre autres, la vérification des contraintes d'intégrité, à l'exception des contraintes de référence et d'identification, qui sont, elles, automatiquement prises en charge. Par exemple, une CI d'existence telle que "le produit commandé ne peut être épuisé" fait porter la vérification sur un autre champ que celui qui lie les relations. Aucun automatisme n'est fourni pour vérifier de telles contraintes; par contre, le langage procédural dispose de primitives permettant de "charger sur lien" l'état d'approvisionnement d'un produit dont le numéro vient d'être introduit. De même, aucun outil non-procédural ne permet de spécifier un champ calculé; il faut encore une fois le faire via le langage procédural.

- le langage procédural dispose de toute une gamme de fonctions de détection d'événements sur les objets, permettant ainsi de pallier par programme à une insuffisance des outils non-procéduraux.

## B Utilisation décisionnelle

Une des grandes forces de 4D est de permettre plusieurs niveaux d'utilisation. On peut distinguer trois niveaux d'utilisation correspondant à trois types d'utilisateurs bien différenciés:

- utilisation en mode "menus créés", où l'outil de développement est totalement fermé à l'utilisateur qui doit se contenter d'exécuter l'application qu'on lui a fournie, exactement comme avec un L3G.

- utilisation en mode "utilisation directe", où des opérations "décisionnelles" sont possibles (requêtes, graphes, rapports, ...), mais où le dictionnaire de données et le langage procédural sont inaccessibles.

- utilisation en mode "structure", où l'outil est totalement disponible pour le développeur (dictionnaire de données, langage procédural, mots de passe...)

La facilité d'utilisation de 4D comme outil décisionnel a d'ailleurs amené ses concepteurs à commercialiser une version bridée de ce produit, limitée à ce type d'utilisation: FileForce, qui correspond en fait à 4D sans langage procédural. Un tel outil est beaucoup mieux adapté à une utilisation décisionnelle que FileMaker.

Seule ombre au niveau de la convivialité: la qualité déplorable des manuels, mal structurés, sans index, renvoyant à des addendum "encore à paraître",... Les primitives du langage procédural sont décrites de façon tellement succincte et imprécise qu'il faut procéder par essais et erreurs avant de comprendre leur fonctionnement. On comprendra aisément qu'un manuel d'utilisation cherche à cacher les faiblesses de l'outil qu'il décrit, mais cela rend doublement difficile la tâche du développeur qui devra:

1. comprendre que la fonctionnalité qu'il recherche n'existe pas
2. pallier à la déficience de l'outil

A la mauvaise qualité de la documentation vient s'ajouter l'absence totale d'aide on-line, due à la méfiance obsessionnelle des concepteurs vis-à-vis des utilisations "illégales"; 4D comporte en outre une sévère protection contre le piratage: la disquette originale est périodiquement demandée.

Les opérations décisionnelles spécifiées ci-dessus ont pu être effectuées sans problème. Les requêtes sont spécifiées très facilement, grâce à un éditeur donnant accès à tous les fichiers liés et à toutes les primitives procédurales. 4D permet de stocker ces requêtes, et de se constituer une librairie fort utile. Les rapports se créent aussi facilement que les formulaires de saisie, avec toutefois le détour habituel par le langage procédural dès qu'un cas sort des possibilités d'utilisation non-procédurales (pour l'accès aux sous-fiches, par exemple). Enfin, 4D permet de dessiner des graphiques sur base de champs de records sélectionnés.

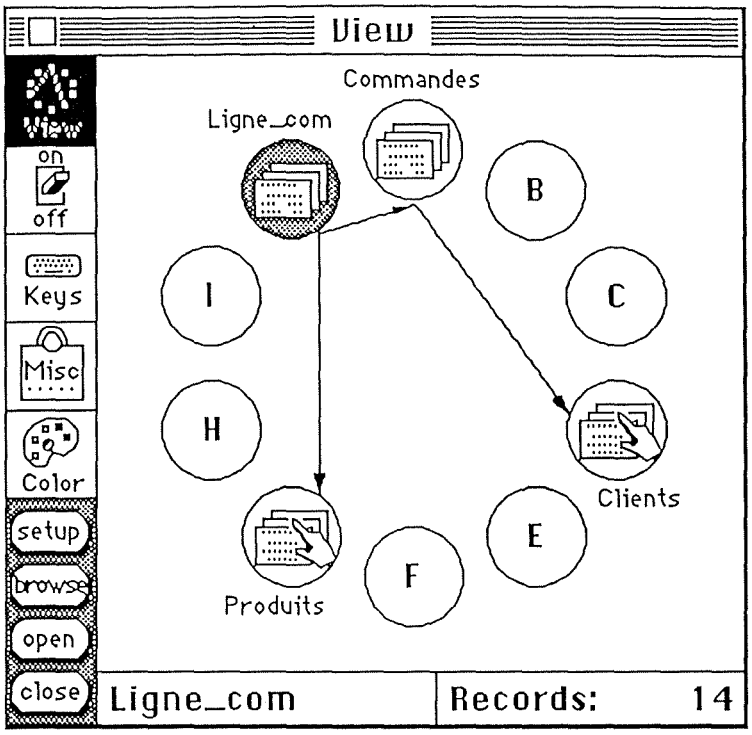
4ème Dimension accepte sans problème les fichiers au format DBase III, après leur transformation au format SYLK par un utilitaire standard du Macintosh: Apple File Exchange. Il accepte également de recevoir des données via un réseau. La version 4.1 permettra en outre de communiquer avec un mainframe via une interface avec SQL Server.

## FoxBase + (version 1.10, 1988)

### A Utilisation opérationnelle

Le mode vue de FoxBase+ correspond à la notion de dictionnaire des données; il centralise autour des relations de la BD les objets du dialogue (formats); les traitements ne sont toutefois pas repris dans ce dictionnaire pour une raison toute simple: le centre de FoxBase+ n'est pas le dictionnaire de données, mais bien les procédures. Cette caractéristique nous a incité à classer cet outil dans les L4G traditionnels

.




*dictionnaire de données de FoxBase+: le mode vue*

## - *productivité*

Nous avons abordé FoxBase+ (FB) après 4ème Dimension (4D), et cela nous a fait perdre beaucoup de temps ! Quittant 4D et son dictionnaire de données central, qui permet de contrôler de bout en bout le développement de l'application, nous sommes entrés dans FB en pensant que son mode "vue", d'aspect très similaire au mode structure de 4D, permettrait le même pilotage du développement. Après de nombreux essais infructueux, nous avons dû perdre tout espoir de pouvoir développer l'application en nous basant sur les outils non-procéduraux.

Pour comprendre cela, il faut se souvenir d'une des caractéristiques fondamentales de FB: il a été conçu, à l'origine, comme un "clône" de DBaseIII+, c'est-à-dire d'un L4G traditionnel puissant mais archaïque, pour lequel un développement d'application ne se conçoit pas sans un travail de programmation classique, à peine moins lourd qu'en L3G. En évoluant, FB a

voulu se conformer à la mode des interfaces conviviales; il a donc superposé à sa couche DBaseIII une couche de dialogue. Mais il ne faut pas s'y tromper: le centre de tout développement sous FB reste toujours la procédure ! A la différence de 4D, où les outils non-procéduraux sont le point de passage obligé, et où les procédures (formules) ne sont que des appendices, les outils non-procéduraux de FB ne sont qu'une représentation déclarative de primitives procédurales. Prenons des exemples précis pour prouver ce que nous avançons:

- FB dispose d'une fenêtre "commande", qui affiche l'historique des opérations effectuées; chaque fois qu'un outil non-procédural est utilisé, cette fenêtre affiche automatiquement son équivalent en commandes procédurales

- l'outil de création de formulaires de saisie transforme automatiquement la description non-procédurale de ce formulaire en code procédural standard

Ce qui nous amène aux conclusions suivantes:

- les outils non-procéduraux de 4D ont une existence autonome au sein de l'outil, même si 4D les transforme sans doute, de façon interne et invisible, en code exécutable.

- les outils non-procéduraux de FB sont en fait des générateurs de code, des outils ponctuels permettant de transformer le plus vite possible les spécifications non-procédurales en procédure, seul objet intéressant vraiment FB.

Cette différence d'approche est plus profonde qu'il n'y paraît de prime abord; elle entraîne les conséquences suivantes:

- FB génère, à partir de spécifications non-procédurales, un code procédural exécutable et modifiable, permettant ainsi au développeur qui modifiera ce code de faire disparaître la cohérence qui existait entre la

description non-procédurale d'un objet et sa représentation interne réelle, avec toutes les confusions que cela peut entraîner lors de la maintenance. Nous ne citons pas cette possibilité de modifier le code dans le seul but de prouver nos dires: elle sera souvent indispensable (voir ci-après).

- FB s'abrite derrière cette possibilité de compléter le code pour proposer des outils non-procéduraux incomplets: aucune contrainte d'intégrité n'est prise en charge automatiquement, la validation d'une saisie restant à charge du développeur, qui devra insérer cette validation dans le code procédural généré à partir de la description du formulaire de saisie. On pourrait citer d'autres exemples montrant les limites des outils non-procéduraux; par exemple, dans un formulaire, l'affichage de champs externes à la relation courante peut se faire par des variables, pourvu qu'elles aient été initialisées, par programme, avant l'appel au formulaire ! Si l'on veut incorporer à un formulaire un menu permettant de choisir un item précis pour un champ "à domaine de valeurs", il faudra, après l'appel au formulaire et la sélection de l'item, construire une structure "case of" pour attribuer réellement au champ la valeur sélectionnée (un menu dans un formulaire n'accepte en effet que des valeurs numériques!). Il en va de même pour les boutons, qui ne déclencheront pas automatiquement une action, mais dont la valeur devra être testée après coup par le programme, qui devra décider lui-même d'effectuer les actions requises.

On voit donc bien que le moteur de FB reste la programmation, et qu'il faut beaucoup programmer pour obtenir avec FB des fonctions qui sont automatiquement prises en charge par 4D.

Cette incomplétude des outils non-procéduraux, cette obligation de devoir systématiquement passer par de la programmation pour pouvoir les utiliser sérieusement, nous incite à les considérer plus comme des accessoires ou des gadgets, qui font certes gagner du temps, mais que l'on ne peut considérer comme de vrais descripteurs d'objets. En 4D, les outils non-procéduraux sont les seuls descripteurs d'objets permis, à l'exclusion de tout code procédural; cela assure la cohérence d'ensemble des objets de l'application, qui peut être

entièrement décrite de façon non-procédurale. En FB, une application ne peut être décrite que sous la forme d'un ensemble de procédures.

Nous avons ainsi découvert qu'il existait une profonde différence de philosophie entre 4D et FB, au-delà de la simple dichotomie L4G traditionnel-L4G avancé que nous avions pressentie. Certes, FB peut être considéré comme un L4G traditionnel: il privilégie bien la procédure comme moyen de pilotage de l'application; mais c'est oublier une de ses caractéristiques essentielles, dont nous n'avons pas encore parlé: la possibilité d'instancier une application générique en application particulière, correspondant à des paramètres précis. FB permet de décrire le canevas général d'une application à l'aide d'un "méta-langage"; ce canevas décrit une application générique, qui sera instanciée en applications réelles en tenant compte de paramètres particuliers tels que les formulaires de saisie, les contraintes d'intégrité,... Cela permet de réutiliser une application dans plusieurs contextes différents, correspondant, par exemple, à plusieurs clients. Ces concepts rapprochent plus FB des outils de génie logiciel que des L4G purs et durs. Pour appuyer cette hypothèse, signalons que FB offre des outils de trace, debugging, paramétrisation d'applications génériques,...

Les contacts que nous avons pris avec un développeur indépendant qui utilise FB comme outil de travail ont confirmé ce que nous venons de dire. Il n'utilise jamais le dictionnaire de données, qu'il considère comme un gadget inutile. Il a développé une "toolbox" lui permettant d'automatiser tout ce que FB ne permettait pas: description des contraintes d'intégrité, des formulaires de saisie,... En deux mots, FoxBase+ devient un outil de développement puissant une fois qu'on lui a fourni les outils d'automatisation qui lui manquaient. Que faut-il en penser, lorsqu'on sait que 4D, lui, permet directement ces automatismes ?

### *- réutilisabilité*

La possibilité qu'offre FB de compiler les applications permet de séparer l'application de l'outil qui l'a générée; l'application pourra tourner sur

plusieurs machines sans devoir pour autant acquérir plusieurs versions de FB, ce qui est impossible en 4D.

Ces caractéristiques destinent FB tout particulièrement aux développeurs indépendants qui apprécieront:

- que le code soit caché
- que leur client ne doive pas acheter FB pour faire tourner l'application vendue
- de pouvoir se constituer une bibliothèque d'applications génériques: une application pour un cabinet médical, qui sera instanciée pour le Docteur X en fonction de ses desideratas précis, une application pour cabinet d'avocats, pour notaires, pour quincaillerie,...
- FB offre même la possibilité de compiler une application de démonstration "bridée" (pas plus de 120 records par relation, ou toute autre limite).

## *- maintenance*

Il est indispensable de bien documenter une application développée sous FB, comme on le ferait en L3G. Le dictionnaire de données (mode "vue") ne reprenant pas dans sa structure les procédures, objets centraux de FB, il faut établir clairement leur hiérarchie d'appel, leurs spécifications,...
En ce qui concerne les formulaires et rapports, on traitera leurs outils de création non-procéduraux comme ce qu'ils sont réellement: de simples générateurs de code qui perdent toute prétention descriptive sur ce code dès qu'il est modifié par le développeur. Il en va de même pour le "générateur" ou "instanciateur" d'applications: dès qu'on modifie le code de l'application générée, on perd la correspondance qui existait entre l'application instanciée et sa description générique. A cet égard, on imaginera aisément la difficulté que représente la maintenance d'un code produit par un générateur automatique... Ces outils générateurs sont donc intéressants pour la maintenance tant qu'on ne touche pas au code qu'ils ont généré; on garde alors la clarté et la facilité de maintenance que permet la description non-procédurale des objets. Hélas, comme on l'a vu, le code doit souvent être "retouché".

## - *performances*

FB compile les applications avant leur exécution, ce qui lui procure un gain significatif pour l'exécution des traitements, calculs, itérations,... (3 fois plus rapide que 4D). En ce qui concerne les opérations "standard" (tri, recherche,...), ses performances sont identiques à 4D: ces routines standard sont déjà optimisées et la compilation n'y change rien. Des essais sur 100000 fiches ont été menés avec FB qui a très bien supporté le choc [SVMA.90.9].

## - *langage procédural*

La cohérence d'approche entre les outils non-procéduraux et les primitives procédurales est totale, puisqu'ils ne font qu'un: l'outil non-procédural ne constitue qu'une représentation "conviviale", un déguisement, de primitives procédurales. La correspondance exacte qui existe entre ces deux types d'outil devrait nous satisfaire, eu égard aux considérations émises plus haut [chapitre 4: critères généraux], à savoir que, tout en permettant d'aborder un problème sous deux angles différents (procédural pour la souplesse et non-procédural pour la productivité), ces deux approches devaient rester cohérentes dans leur façon d'aborder les objets de l'application. Or, cette satisfaction est toute relative puisque FB ne nous offre pas à proprement parler la possibilité d'aborder un problème sous un angle purement non-procédural. Il nous offre la souplesse, mais pas la productivité qu'on est en droit d'attendre d'un bon L4G.

Néanmoins, il ne faut pas trop noircir le tableau: le langage procédural de FB est très puissant et facilite le développement d'applications, à partir du moment où on a accepté que tout devait se faire par programmation, et qu'il ne fallait pas attendre monts et merveilles des outils non-procéduraux.

En conclusion, si l'on considère FB comme un L4G, censé améliorer la productivité en rapprochant la programmation des spécifications de l'analyse

fonctionnelle, on le trouvera médiocre: outils non-procéduraux incomplets, peu d'automatismes,... En revanche, si on le considère comme un outil de génie logiciel, censé améliorer la productivité en privilégiant la réutilisabilité des applications, on le verra d'un oeil moins sévère.

### B Utilisation décisionnelle

L'utilisation décisionnelle n'est pas la vocation première de FB, loin s'en faut ! Cet outil est de trop bas niveau, trop proche des aspects techniques, pour qu'un utilisateur final y soit à l'aise. Par exemple, il ne suffit pas, comme en 4D, de dire que tel champ d'une relation est indexé pour que cet index soit automatiquement utilisé: il faudra l'ouvrir explicitement. Un utilisateur final pourra donc facilement effectuer une sélection sur 10000 fiches non-indexées, ce qui le "refroidira" sérieusement vis-à-vis de l'outil. Nous rappelons qu'un L4G décisionnel doit au contraire s'efforcer de rassurer l'utilisateur final...

Seul aspect positif: une aide on-line complète et "sensible au contexte", c'est à dire qu'elle donnera des explications spécifiques à l'opération que l'utilisateur est en train d'accomplir. Cette aide est centrée sur les primitives procédurales du langage, au même titre que l'ensemble de l'outil.

Toutes les opérations décisionnelles spécifiées au début de ce chapitre ont pu être effectuées (sauf l'affichage de graphiques), mais au prix d'un gros travail de programmation incompatible avec ce que nous entendons par "utilisation décisionnelle".

FoxBase+ accepte sans transformation préalable les fichiers et programmes au format DBase III. Nous avons pu importer une application opérationnelle complète depuis DBaseIII+, et la faire tourner sans problème. Il accepte également de recevoir des données via un réseau; grâce à sa compatibilité avec DBase, il peut coopérer directement avec les PC connectés au même réseau.

## 6.5 Conclusions de l'expérimentation

FoxBase+ (FB) et 4ème dimension (4D) ont des approches différentes du développement d'applications:

- 4D est un vrai L4G, allégeant au maximum la tâche du développeur d'applications uniques. Il privilégie la convivialité, la productivité, la non-procéduralité. En revanche, la dispersion des traitements et de la dynamique de l'application au sein des objets qui la composent enchaîne irrémédiablement l'application à un seul contexte ou scénario d'utilisation.

- FB est un outil de génie logiciel, qui voit la tâche de développement comme un processus de production industriel, où on sculpte d'abord les moules qui serviront ensuite à fabriquer en série des applications paramétrées, adaptées à des contextes d'utilisation différents. Il privilégie la réutilisabilité des applications, en permettant de les développer à un haut niveau de généralité, indépendamment de tout contexte d'utilisation trop précis.

En résumé: si l'on doit développer une application unique, spécifique, on choisira 4D qui permettra un développement beaucoup plus rapide que FB. En revanche, si doit développer un ensemble d'applications basées sur un même "thème", ou si l'on veut préserver un certain niveau de réutilisabilité à une application complexe, on choisira FB, et on acceptera de consacrer plus de temps à la construction d'un canevas, pour pouvoir ensuite l'adapter facilement à toute une gamme d'environnements d'utilisation.

Ces outils appartiennent donc bien à deux "lignées" toujours opposées actuellement; elles ont le même but: améliorer la productivité du développement, mais proposent des moyens différents pour y arriver:

- la lignée des L4G vise la productivité immédiate. Elle préconise de rapprocher l'outil des spécifications de l'application, afin d'automatiser le développement, au détriment de la réutilisabilité et de la productivité à long terme.

- la lignée des outils de génie logiciel vise la productivité à long terme. Elle préconise de favoriser la réutilisabilité des applications, au détriment du temps de développement et de la productivité immédiate.

Le débat déborde sur les méthodes d'analyse, puisque, comme nous l'avons dit, les outils de développement actuels offrent des approches propres à révolutionner ces méthodes conçues pour les anciens L3G. La polémique porte sur l'étape de conception: faut-il automatiser au maximum la transformation des spécifications fonctionnelles en code exécutable, et donc tenter de faire disparaître cette étape ? Faut-il au contraire la maintenir, et y procéder à un redécoupage soigneux des fonctions de l'analyse fonctionnelle en une hiérarchie de modules, de façon à favoriser au maximum la réutilisabilité de ces modules?

Le débat est loin d'être clos... On objectera toutefois que le temps passé à effectuer une conception soigneuse doit servir à quelque chose: il nous semble inutile de consacrer tout ce temps à structurer, généraliser, bref, à rendre *portable* une application qui ne sera jamais effectivement *portée* dans un autre contexte d'utilisation... D'autre part, la facilité de développement accrue que permettent les L4G amoindrit beaucoup les avantages qu'on peut retirer de la réutilisabilité: pourquoi perdre son temps à construire une application réutilisable, modulaire, paramétrée, quand les L4G permettent de développer très rapidement chaque application particulière ?

# 7.Conclusions

## 1. Les limites des L4G

La non-procéduralité est l'atout principal d'un L4G dans sa course vers une meilleure productivité. Mais ce **gain de productivité** n'est obtenu qu'au prix d'une restriction du domaine d'application et d'une **perte de flexibilité.**

*"Chaque fois qu'on s'éloigne d'un niveau d'abstraction de la façon dont pense un processeur (binaire, registres, transferts mémoire,...), on perd contrôle sur les opérations de ce processeur. C'est vrai à chaque évolution de langage et c'est une conséquence inévitable de la capacité de travailler en termes des problèmes d'une application, et non des opérations d'un processeur."* [RAPA.87]

Un outil non-procédural ne peut prétendre à la souplesse d'un langage procédural, sous peine de devenir aussi complexe à manipuler que lui. Le manque de souplesse et la restriction du champ d'application sont le prix à payer si l'on veut la facilité d'emploi et la productivité; on ne peut avoir le beurre et l'argent du beurre...

Les L4G qui veulent rendre possible le développement d'applications complexes doivent donc fournir un langage procédural permettant de reprendre le contrôle des opérations, au cas ou les outils non-procéduraux déclareraient forfait.

Prenons un exemple : je veux allumer une cigarette.

Si je dois le demander à un robot piloté de façon procédurale, je devrai lui spécifier avec précision la séquence d'opérations élémentaires à accomplir:

- prendre la boîte d'allumettes
- l'ouvrir
- prendre une allumette
- la frotter contre la boîte
- si l'allumette a pris, allumer la cigarette
- si l'allumette casse, en prendre une autre et recommencer

Si je dispose d'un robot non-procédural, qui donc aura mémorisé la séquence d'instructions donnée ci-dessus, je lui demanderai simplement de m'allumer une cigarette.

Mais comment réagira-t-il si l'allumette saisie est pliée, fragile, mal enduite de soufre ? Il faudra que je le reprenne sous contrôle pour lui spécifier de façon procédurale ce qu'il doit faire: saisir l'allumette plus près du soufre, la frotter délicatement,...

C'est exactement ce qui se passe avec les L4G. En développant une application complexe, on tombe tôt ou tard sur des cas particuliers, devant lesquels les outils non-procéduraux abandonnent. On ne peut exiger d'un L4G qu'il prévienne complètement ce genre de situation; il faut néanmoins qu'elles soient rares, accidentelles, et que, le cas échéant, il soit possible de les résoudre sans perdre trop de temps. En effet, cette navette incessante entre outils non-procéduraux et langage procédural est un des aspects les plus désagréables des L4G. Ils nous habituent à une grande facilité d'utilisation, nous font oublier les difficultés de la programmation, pour brusquement nous réveiller aux dures réalités lorsqu'un cas difficile surgit. Plus encore que la qualité de leurs langages procéduraux (qui s'améliore), ce sont ces ruptures entre non-procéduralité et procéduralité qui rendent l'utilisation des L4G parfois déroutante et fatigante. En Cobol, la façon d'aborder un problème est toujours identique: il faut le spécifier pas à pas, on sait à quoi s'en tenir. Avec un L4G, tout se passe bien, comme si la programmation était devenue un jeu d'enfant, puis, brutalement, les outils non-procéduraux vous laissent tomber; vous passez de l'autre côté du miroir, dans une réalité qui vous était masquée jusque là, et à

laquelle vous ne comprenez pas grand chose: un simple trait tiré entre deux relations apparaît, vu sous cet angle, d'une énorme complexité.

Un bon L4G essayera donc de réduire au maximum le recours à son langage procédural; ces recours étant malgré tout inévitables, le langage procédural devra être cohérent avec les outils non-procéduraux dans sa façon d'aborder les objets de l'application, afin de limiter la perte de temps.

On doit néanmoins être conscient que la productivité s'obtient au détriment de la flexibilité, et que toute amélioration de la flexibilité (en recourant au langage procédural du L4G) entraîne une perte de productivité.

Une autre limite des L4G tient à l'intégration des objets et fonctions qu'ils offrent. Cette intégration des objets (BD, dialogue, traitements, pilotage) permet de centraliser le développement d'une application dans une structure homogène et cohérente: le dictionnaire des données. L'éclatement des traitements et de la dynamique, l'absence de modularité et la forte cohésion qui en résultent rendent l'application ainsi développée fort dépendante du contexte d'utilisation prévu: on aura beaucoup de mal à faire tourner l'application selon un autre scénario. L'intégration des L4G produit donc des applications difficilement portables et réutilisables.

Les limites des L4G peuvent se synthétiser en deux antagonismes:

| |
|---|
| productivité (non-procédural) <-> flexibilité (procédural) |
| intégration <-> réutilisabilité |

On retrouve ici les deux grandes solutions proposées à la crise du développement logiciel:

-les L4G, qui favorisent la productivité immédiate (non-procéduralité, intégration), au détriment de la productivité à long terme (réutilisabilité) et de la flexibilité.

-les outils de génie logiciel, qui favorisent la productivité à long terme (réutilisabilité) et la flexibilité, au détriment de la productivité immédiate.

## 2. Perspectives

La grande tendance dans l'évolution des générations de langages est de proposer des outils au niveau non-procédural de plus en plus élevé, se rapprochant du mode de pensée humain. Cette évolution est continue: il serait hasardeux de prétendre que tel L4G est procédural, et que tel autre ne l'est plus; il n'existe pas de rupture brutale entre ces deux concepts. Par certains aspects, les L3G étaient déjà non-procéduraux: une addition en Pascal (a := b + c) se résume à la spécification déclarative du but à atteindre; on est loin de l'assembleur, avec lequel il fallait spécifier pas à pas le détail des opérations: (LDA b, ADD c, STA a). Plutôt que de classer les langages en deux groupes : les procéduraux et les non-procéduraux, il faudrait les répartir selon une échelle de non-procéduralité, allant de pair avec une spécialisation croissante. Plus un outil est non-procédural, plus le niveau des objectifs à lui spécifier est élevé et plus l'outil devra faire preuve de jugement quant aux sous-objectifs de mise en oeuvre. Un outil non-procédural dépourvu de toute capacité de jugement ne peut effectuer qu'une tâche précise, en dehors de laquelle il sera inutilisable à moins de le re-programmer. Les L4G offrent des outils non-procéduraux capables d'exercer un jugement rudimentaire: utiliser un outil dans tel contexte implique que l'utilisateur vise tel objectif. Le degré supérieur n'est atteint que par l'intelligence artificielle, qui offre des outils capables de se satisfaire d'objectifs imprécis, de niveau conceptuel, mais c'est au prix d'une spécialisation extrême: un système expert ne fonctionne que dans un domaine d'expertise très restreint.

On distingue en effet **trois niveaux d'abstraction** [BOUT.84]:

- le niveau **conceptuel**, décisionnel, est celui de la représentation naturelle qu'a l'homme du réel perçu, des problèmes posés et des voies de solution à suivre. Les objectifs de ce niveau sont synthétiques, globaux, éloignés des préoccupations de mise en oeuvre pratique. Ils sont souvent imprécis, peu structurés, peu cohérents.

- le niveau **logique**, est plus travaillé, plus structuré et rationnel. C'est à ce niveau qu'une mission conceptuelle se concrétise en une structure cohérente, en un modèle d'action structuré et cohérent, mais également abstrait et théorique.

- le niveau **physique**, opérationnel, technique, est soumis à l'ensemble des contraintes de fonctionnement des machines réelles; il met en mouvement des automates physiques. Il exécute mécaniquement le mieux possible le système décrit au niveau logique pour résoudre les problèmes posés au niveau conceptuel.

Le niveau physique correspond aux langages de bas niveau, très techniques et procéduraux; grosso modo, les trois premières générations.

Le niveau logique correspond à la structuration abstraite des objectifs conceptuels; l'analyse fonctionnelle, les spécifications de haut niveau, que les L4G tentent de prendre en charge directement.

Le niveau conceptuel correspond aux L5G, qui permettent de prendre en charge les missions générales de l'utilisateur sans qu'il doive les traduire en spécifications structurées.

Le renforcement des capacités de jugement des L4G nous semblerait utile; un système expert modélisant une méthode de développement adaptée permettrait de pallier au manque de cohérence des L4G (éclatement des traitements et de la dynamique), et rendrait peut-être moins utopique le concept de "programmation sans programmeurs". Encore une fois, il faut être conscient de la restriction supplémentaire du champ d'application que cela entraînerait, mais cela correspondrait bien à l'objectif principal des L4G: la productivité

maximale, même au prix d'une perte de flexibilité. Toutefois, un système expert d'aide au développement ne pourra être envisagé que lorsqu'une méthode de développement parfaitement adaptée aux L4G sera mise au point; on s'est aperçu à de nombreuses reprises qu'on était loin du compte !

## 3. Evaluation de la méthode

Tant que l'offre en matière de L4G sera aussi disparate et peu standardisée, l'utilisation d'une méthode de sélection rigoureuse nous semble indispensable.

Il faudra être conscient, lors du choix d'un L4G, que d'autres approches sont envisageables: les progiciels, les outils de génie logiciel sont parfois plus adaptés aux besoins de l'organisation. On gardera toujours à l'esprit que, si un L4G améliore la productivité immédiate du développement (non-procéduralité et intégration), il ne le permet qu'au prix d'une dégradation de la flexibilité et de la réutilisabilité pouvant, à terme, pénaliser les capacités du CTI. A cela s'ajoute le problème de la maintenance qui, tant qu'une méthode de développement adaptée n'est pas proposée, nécessitera grosso modo le même effort qu'avec les L3G (importance de la documentation,...).

La méthode proposée ici permet en outre de prendre conscience de l'ensemble des besoins de l'organisation en matière de L4G; elle pourra ainsi corriger les imprécisions du schéma directeur à cet égard.

Nous sommes toutefois conscients qu'il manque à cette méthode une évaluation pratique, dans un cadre organisationnel concret.

# BIBLIOGRAPHIE


[BODA.89.a] F. Bodart et Y. Pigneur,
Conception assistée des applications informatiques, Tome 1
Etude d'opportunité et analyse conceptuelle, Masson, 1989

[BODA.89.b] F. Bodart
Notes du cours d'Interface homme-machine,
Institut d'Informatique, FUNDP, Namur, 1989.

[BODA.90] F. Bodart
Notes du cours de Gestion stratégique des ressources informatiques,
Institut d'Informatique, FUNDP, Namur, 1990.

[BOUT.84] J.S. Boutel et J.P. Pignon,
Problématique et typologie des nouveaux langages de l'informatique,
01 Informatique, n° 178, avril 1984;
Concepts fondamentaux des nouveaux langages de l'informatique,
01 Informatique, n° 180, juin-juillet 1984;
Langages de quatrièmes génération: les produits disponibles,
01 Informatique, n° 183, septembre-octobre 1984;
Langages de quatrième génération: une tentative de classification,
01 Informatique, n° 184, novembre 1984.

[Brochures fournisseurs] :
Computer Associates (CA-Universe, Super DB), McCormack & Dodge Corporation (Millenium Application), Applied Data Research (ADR/ Ideal), Cincom Systems (Supra/Mantis), Mimer Information Systems, Uniface, Information Builders (Focus), Cullinet (IDMS/R), Oracle Corporation, Sybase inc., Relational Database Systems inc. (Informix)

[BYTE.1989] Byte
Database trends, septembre 1989

[COMP.90] M. Sullivan-Trainor
Buyers' scorecard: Sybase wins by a nose,
Computerworld, vol. XXIV, n° 10, mars 1990

[FEGE.84] A.R. Feuer, N.H. Gehani,
A methodology for comparing & assessing programming languages: Ada, C, Pascal.
Prentice-Hall, 1984

[GCM.86] Générale des Carrières et des Mines - Exploitation;
département informatique & télécommunications, division études techniques
Evaluation d'un outil de quatrième génération pour l'IBM 4381; mai 86

[HAIN.88] J.L. Hainaut
Les environnements de quatrième génération
Journ'Almin, n° 8, septembre 1988

[HAIN.86] J.L. Hainaut
Conception assistée des applications informatiques, tome II
Conception de la base de données, Masson, 1986

[HENN.87] T. Hennaux et P. Smets
Les outils de quatrième génération: concepts et enquêtes
Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, 1987

[JALI.89] P. Jalics, M. Matos
An experimental analysis of the performance of fourth generation tools on PC.
Communications of the ACM, vol. 32, n° 11, novembre 1989

[LAUR.89] D. Laurent et V. Léonard
Aspects méthodologiques des environnements de quatrième génération.
Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, 1989

[LESU.89.a] R. Lesuisse

Notes du cours de Théorie des organisations, matière approfondie
Institut d'Informatique, FUNDP, Namur, 1989

[LESU.89.b] R. Lesuisse
Notes du cours Bureautique,
Institut d'Informatique, FUNDP, Namur, 1989

[MART.85] J. Martin
Fourth generation languages, vol. 1: Principles
Prentice-Hall, 1985

[MART.86.a] J. Martin
Fourth generation languages, vol. 2: representative 4GLs
Prentice-Hall, 1986

[MART.86.b] J. Martin
Fourth generation languages, vol. 3: 4GLs from IBM
Prentice-Hall, 1986

[MISR.88] S.K. Misra et P.J. Jalics
Third-generation versus fourth-generation software developpement
IEEE Software, juillet 1988

[MOUL.90]
Notes du cours de systèmes interactifs d'aide à la décision,
Institut d'Informatique, FUNDP, Namur, 1990

[RAPA.87] M. Rapaport
Designing with databases; fourth generation languages
Computer language, octobre 1987

[ROUX.88] F.G. Roux
Classification des fonctions des langages de quatrième génération
L'Informatique Professionnelle, mars 1988

[SHNE.87] B. Shneiderman

Designing the user interface: strategies for effective human-computer interaction.
Addison-Wesley, 1987

[SUCH.88] W. Suchun
Comparaison d'un SGBD Codasyl et d'un SGBD relationnel
Mémoire de fin d'études, Institut d'Informatique, FUNDP, Namur, 1988

[SVM.yy.nn] Science&Vie Micro
nn= numéro; yy=année
n° 52, 55, 58, 61, 62, 63, 73, hors série logiciels PC

[SVMA.yy.nn] Science&Vie Macintosh
nn= numéro; yy=année
n° 9, 12

[VERN.88] J. Verner, G. Tate
Estimating size and effort in fourth-generation development
IEEE Software, juillet 1988